6

Monitoring the Database

It is important that you monitor the operation of your database on a regular basis. Doing so not only informs you of errors that have not yet come to your attention but also gives you a better understanding of the normal operation of your database. Being familiar with normal behavior in turn helps you recognize when something is wrong.

> **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.

- Monitoring Errors and Alerts
  You can monitor database errors and alerts to prevent, detect, and solve problems.
- Monitoring Performance
  Monitoring performance includes monitoring locks and wait events and querying a set of data dictionary views.
- Monitoring Quarantined Objects
  Object quarantine enables an Oracle database to function even when there are corrupted, unrecoverable objects. The `V$QUARANTINE` view contains information about quarantined objects.
- Automatically Monitoring Schema Objects
  Oracle Database can automatically track the activities and usage of certain schema objects, such as tables and materialized views.

6.1 Monitoring Errors and Alerts

You can monitor database errors and alerts to prevent, detect, and solve problems.

> **Note:**
>
> The easiest and best way to monitor the database for errors and alerts is with the Database Home page in Oracle Enterprise Manager Cloud Control (Cloud Control). See the Cloud Control online help for more information. This section provides alternate methods for monitoring, using data dictionary views, PL/SQL packages, and other command-line facilities.

- Monitoring Errors with Trace Files and the Alert Log
  A trace file is a file that contains diagnostic data used to investigate problems. An alert log is a file that provides a chronological log of database messages and errors.

- Monitoring a Database with Server-Generated Alerts
  A server-generated alert is a notification from the Oracle Database server of an impending problem.

6.1.1 Monitoring Errors with Trace Files and the Alert Log

A trace file is a file that contains diagnostic data used to investigate problems. An alert log is a file that provides a chronological log of database messages and errors.

- About Monitoring Errors with Trace Files and the Alert Log
  The trace file and alert log contain information about errors.
- Controlling the Size of an Alert Log
  To control the size of an alert log, change the segment size and number of segments used by the alert log.
- Controlling the Size of Trace Files
  You can control the maximum size of all trace files (excluding the alert log) using the initialization parameter `MAX_DUMP_FILE_SIZE`.
- Controlling When Oracle Database Writes to Trace Files
  Background processes always write to a trace file when appropriate.
- Reading the Trace File for Shared Server Sessions
  If shared server is enabled, each session using a dispatcher is routed to a shared server process, and trace information is written to the server trace file only if the session has enabled tracing (or if an error is encountered). Therefore, to track tracing for a specific session that connects using a dispatcher, you might have to explore several shared server trace files.

6.1.1.1 About Monitoring Errors with Trace Files and the Alert Log

The trace file and alert log contain information about errors.

Each server and background process can write to an associated **trace file**. When an internal error is detected by a process, it dumps information about the error to its trace file. Some of the information written to a trace file is intended for the database administrator, and other information is for Oracle Support Services. Trace file information is also used to tune applications and instances.

> **Note:**
>
> Critical errors also create incidents and incident dumps in the Automatic Diagnostic Repository. See Diagnosing and Resolving Problems for more information.

The **alert log** is a chronological log of messages and errors, and includes the following items:

- All internal errors (`ORA-00600`), block corruption errors (`ORA-01578`), and deadlock errors (`ORA-00060`) that occur
- Administrative operations, such as some `CREATE`, `ALTER`, and `DROP` statements and `STARTUP`, `SHUTDOWN`, and `ARCHIVELOG` statements
- Messages and errors relating to the functions of shared server and dispatcher processes
- Errors occurring during the automatic refresh of a materialized view

- The values of all initialization parameters that had nondefault values at the time the database and instance start

Oracle Database uses the alert log to record these operations as an alternative to displaying the information on an operator's console (although some systems also display information on the console). If an operation is successful, a "completed" message is written in the alert log, along with a timestamp.

The alert log is maintained as both an XML-formatted file and a text-formatted file. You can view either format of the alert log with any text editor or you can use the ADRCI utility to view the XML-formatted version of the file with the XML tags stripped.

Check the alert log and trace files of an instance periodically to learn whether the background processes have encountered errors. For example, when the log writer process (LGWR) cannot write to a member of a log group, an error message indicating the nature of the problem is written to the LGWR trace file and the alert log. Such an error message means that a media or I/O problem has occurred and should be corrected immediately.

Oracle Database also writes values of initialization parameters to the alert log, in addition to other important statistics.

The alert log and all trace files for background and server processes are written to the Automatic Diagnostic Repository, the location of which is specified by the `DIAGNOSTIC_DEST` initialization parameter. The names of trace files are operating system specific, but each file usually includes the name of the process writing the file (such as LGWR and RECO).

See Also:

- "Diagnosing and Resolving Problems" for information about the Automatic Diagnostic Repository (ADR).
- "Alert Log" for additional information about the alert log.
- "Viewing the Alert Log"
- *Oracle Database Utilities* for information on the ADRCI utility.
- Your operating system specific Oracle documentation for information about the names of trace files

6.1.1.2 Controlling the Size of an Alert Log

To control the size of an alert log, change the segment size and number of segments used by the alert log.

By default, the alert log consists of 20 segments with each being 50 Mb in size. The alert log has segmentation roataion enabled, which means that when all alert log files are full, Oracle will overwrite the files beginning with the oldest file.

To control the size of an alert log:

- Back up any alert log files you wish to keep.
- Delete the alert log file or files.

6.1.1.3 Controlling the Size of Trace Files

You can control the maximum size of all trace files (excluding the alert log) using the initialization parameter `MAX_DUMP_FILE_SIZE`.

You can set this parameter in the following ways:

- A numerical value specifies the maximum size in operating system blocks. The specified value is multiplied by the block size to obtain the limit.
- A number followed by a K, M, or G suffix specifies the file size in kilobytes, megabytes, or gigabytes. The default value is 32M on Oracle Database Free, and 1G on all other Oracle Database offerings.
- `UNLIMITED`, which specifies no limit.

- Trace File Segmentation and MAX_DUMP_FILE_SIZE
  Oracle Database can automatically segment trace files based on the limit you specify with the `MAX_DUMP_FILE_SIZE` initialization parameter. When a limit is reached, the database renames the current trace file using a sequential number, and creates an empty file with the original name.

> **See Also:**
>
> - *Oracle Database Reference* for more information about the `MAX_DUMP_FILE_SIZE` initialization parameter.
> - About the Oracle Database Fault Diagnosability Infrastructure for more information about IPS.

6.1.1.3.1 Trace File Segmentation and MAX_DUMP_FILE_SIZE

Oracle Database can automatically segment trace files based on the limit you specify with the `MAX_DUMP_FILE_SIZE` initialization parameter. When a limit is reached, the database renames the current trace file using a sequential number, and creates an empty file with the original name.

The following table describes how trace files are segmented based on the `MAX_DUMP_FILE_SIZE` setting.

Table 6-1 The MAX_DUMP_FILE_SIZE Parameter and Trace File Segmentation

MAX_DUMP_FILE_SIZE Setting	Trace File Segmentation
`UNLIMITED`	Trace files are not segmented.
Larger than `25M`	Trace files are segmented on a boundary that is 1/5 of the `MAX_DUMP_FILE_SIZE` setting. Trace files with sizes that are less than this boundary in size are not segmented. For example, if the `MAX_DUMP_FILE_SIZE` setting is `100M`, then the boundary is 20 MB (1/5 of 100 MB).
`25M` or less	Trace files are not segmented.

There can be up to five segments, but the total combined size of the segments cannot exceed the `MAX_DUMP_FILE_SIZE` limit. When the combined size of all segments of the trace file exceeds the specified limit, the oldest segment after the first segment is deleted, and a new, empty segment is created. Therefore, the trace file always contains the most recent trace information. The first segment is not deleted because it might contain relevant information about the initial state of the process.

Segmentation improves space management for trace files. Specifically, segmentation enables you to manage trace files in the following ways:

- You can purge old trace files when they are no longer needed.
- You can diagnose problems with smaller trace files and isolate trace files that must be packaged for the incident packaging service (IPS).

> **Note:**
>
> Any segment that covers a time range that includes an incident is not deleted. It is kept in addition to the five default segments.

6.1.1.4 Controlling When Oracle Database Writes to Trace Files

Background processes always write to a trace file when appropriate.

In the case of the ARC*n* background process, it is possible, through the `LOG_ARCHIVE_TRACE` initialization parameter, to control the amount and type of trace information that is produced. To do so:

- Follow the instructions described in the section "Controlling Trace Output Generated by the Archivelog Process".

Other background processes do not have this flexibility.

Trace files are written on behalf of server processes whenever critical errors occur. Additionally, setting the initialization parameter `SQL_TRACE = TRUE` causes the SQL trace facility to generate performance statistics for the processing of all SQL statements for an instance and write them to the Automatic Diagnostic Repository.

Optionally, you can request that trace files be generated for server processes. Regardless of the current value of the `SQL_TRACE` initialization parameter, each session can enable or disable trace logging on behalf of the associated server process by using the SQL statement `ALTER SESSION SET SQL_TRACE`. This example enables the SQL trace facility for a specific session:

```
ALTER SESSION SET SQL_TRACE TRUE;
```

Use the `DBMS_SESSION` or the `DBMS_MONITOR` packages to control SQL tracing for a session.

> **Note:**
>
> The SQL trace facility for server processes can cause significant system overhead resulting in severe performance impact, so you should enable this feature only when collecting statistics.

> **See Also:**
>
> - "Diagnosing and Resolving Problems" for more information about how the database handles critical errors, otherwise known as *incidents*.

6.1.1.5 Reading the Trace File for Shared Server Sessions

If shared server is enabled, each session using a dispatcher is routed to a shared server process, and trace information is written to the server trace file only if the session has enabled tracing (or if an error is encountered). Therefore, to track tracing for a specific session that connects using a dispatcher, you might have to explore several shared server trace files.

To help you, Oracle provides a command line utility program, `trcsess`, which consolidates all trace information pertaining to a user session in one place and orders the information by time.

> **See Also:**
>
> *Oracle Database SQL Tuning Guide* for information about using the SQL trace facility and using `TKPROF` and `trcsess` to interpret the generated trace files

6.1.2 Monitoring a Database with Server-Generated Alerts

A server-generated alert is a notification from the Oracle Database server of an impending problem.

- About Monitoring a Database with Server-Generated Alerts
  A server-generated alert may contain suggestions for correcting the problem. Notifications are also provided when the problem condition has been cleared.
- Setting and Retrieving Thresholds for Server-Generated Alerts
  You can view and change threshold settings for the server alert metrics using the `SET_THRESHOLD` and `GET_THRESHOLD` procedures of the `DBMS_SERVER_ALERT` PL/SQL package.
- Viewing Server-Generated Alerts
  The easiest way to view server-generated alerts is by accessing the Database Home page of Cloud Control, but there are other methods of viewing these alerts.
- Server-Generated Alerts Data Dictionary Views
  You can query data dictionary views for information about server-generated alerts.

6.1.2.1 About Monitoring a Database with Server-Generated Alerts

A server-generated alert may contain suggestions for correcting the problem. Notifications are also provided when the problem condition has been cleared.

Alerts are automatically generated when a problem occurs or when data does not match expected values for metrics, such as the following:

- Physical Reads Per Second
- User Commits Per Second

- SQL Service Response Time

Server-generated alerts can be based on threshold levels or can issue simply because an event has occurred. Threshold-based alerts can be triggered at both threshold warning and critical levels. The value of these levels can be customer-defined or internal values, and some alerts have default threshold levels which you can change if appropriate. For example, by default a server-generated alert is generated for tablespace space usage when the percentage of space usage exceeds either the 85% warning or 97% critical threshold level. Examples of alerts not based on threshold levels are:

- `Snapshot Too Old`
- `Resumable Session Suspended`
- `Recovery Area Space Usage`

An alert message is sent to the predefined persistent queue `ALERT_QUE` owned by the user `SYS`. Cloud Control reads this queue and provides notifications about outstanding server alerts, and sometimes suggests actions for correcting the problem. The alerts are displayed on the Cloud Control Database Home page and can be configured to send email or pager notifications to selected administrators. If an alert cannot be written to the alert queue, a message about the alert is written to the Oracle Database alert log.

Background processes periodically flush the data to the Automatic Workload Repository to capture a history of metric values. The alert history table and `ALERT_QUE` are purged automatically by the system at regular intervals.

6.1.2.2 Setting and Retrieving Thresholds for Server-Generated Alerts

You can view and change threshold settings for the server alert metrics using the `SET_THRESHOLD` and `GET_THRESHOLD` procedures of the `DBMS_SERVER_ALERT` PL/SQL package.

> **Note:**
>
> The most convenient way to set and retrieve threshold values is to use the graphical interface of Cloud Control. See the Cloud Control online help about managing alerts for instructions.

- Setting Threshold Levels
  The `SET_THRESHOLD` procedure in the `DBMS_SERVER_ALERT` package can set threshold levels.
- Retrieving Threshold Information
  The `GET_THRESHOLD` procedure in the `DBMS_SERVER_ALERT` package can retrieve threshold information.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_SERVER_ALERT` package

6.1.2.2.1 Setting Threshold Levels

The `SET_THRESHOLD` procedure in the `DBMS_SERVER_ALERT` package can set threshold levels.

To set threshold levels:

- Run `SET_THRESHOLD` procedure in the `DBMS_SERVER_ALERT` package, and specify the appropriate arguments.

The following example shows how to set thresholds with the `SET_THRESHOLD` procedure for CPU time for each user call for an instance:

```
DBMS_SERVER_ALERT.SET_THRESHOLD(
 DBMS_SERVER_ALERT.CPU_TIME_PER_CALL, DBMS_SERVER_ALERT.OPERATOR_GE, '8000',
 DBMS_SERVER_ALERT.OPERATOR_GE, '10000', 1, 2, 'inst1',
 DBMS_SERVER_ALERT.OBJECT_TYPE_SERVICE, 'main.regress.rdbms.dev.us.example.com');
```

In this example, a warning alert is issued when CPU time exceeds 8000 microseconds for each user call and a critical alert is issued when CPU time exceeds 10,000 microseconds for each user call. The arguments include:

- `CPU_TIME_PER_CALL` specifies the metric identifier. For a list of support metrics, see *Oracle Database PL/SQL Packages and Types Reference*.
- The observation period is set to 1 minute. This period specifies the number of minutes that the condition must deviate from the threshold value before the alert is issued.
- The number of consecutive occurrences is set to 2. This number specifies how many times the metric value must violate the threshold values before the alert is generated.
- The name of the instance is set to `inst1`.
- The constant `DBMS_ALERT.OBJECT_TYPE_SERVICE` specifies the object type on which the threshold is set. In this example, the service name is `main.regress.rdbms.dev.us.example.com`.

6.1.2.2.2 Retrieving Threshold Information

The `GET_THRESHOLD` procedure in the `DBMS_SERVER_ALERT` package can retrieve threshold information.

To retrieve threshold values:

- Run the `GET_THRESHOLD` procedure in the `DBMS_SERVER_ALERT` package and specify the appropriate arguments.

The following example retrieves threshold values:

```
DECLARE
 warning_operator         BINARY_INTEGER;
 warning_value            VARCHAR2(60);
 critical_operator        BINARY_INTEGER;
 critical_value           VARCHAR2(60);
 observation_period       BINARY_INTEGER;
 consecutive_occurrences  BINARY_INTEGER;
BEGIN
 DBMS_SERVER_ALERT.GET_THRESHOLD(
 DBMS_SERVER_ALERT.CPU_TIME_PER_CALL, warning_operator, warning_value,
    critical_operator, critical_value, observation_period,
    consecutive_occurrences, 'inst1',
 DBMS_SERVER_ALERT.OBJECT_TYPE_SERVICE, 'main.regress.rdbms.dev.us.example.com');
 DBMS_OUTPUT.PUT_LINE('Warning operator:       ' || warning_operator);
```

```
 DBMS_OUTPUT.PUT_LINE('Warning value:          ' || warning_value);
 DBMS_OUTPUT.PUT_LINE('Critical operator:      ' || critical_operator);
 DBMS_OUTPUT.PUT_LINE('Critical value:         ' || critical_value);
 DBMS_OUTPUT.PUT_LINE('Observation_period:     ' || observation_period);
 DBMS_OUTPUT.PUT_LINE('Consecutive occurrences:' || consecutive_occurrences);
END;
/
```

You can also check specific threshold settings with the `DBA_THRESHOLDS` view. For example:

```
SELECT metrics_name, warning_value, critical_value, consecutive_occurrences
   FROM DBA_THRESHOLDS
   WHERE metrics_name LIKE '%CPU Time%';
```

6.1.2.3 Viewing Server-Generated Alerts

The easiest way to view server-generated alerts is by accessing the Database Home page of Cloud Control, but there are other methods of viewing these alerts.

If you use your own tool rather than Cloud Control to display alerts, then complete the following steps to view server-generated alerts:

1. Subscribe to the `ALERT_QUE`.
2. Read the `ALERT_QUE`.
3. Display an alert notification after setting the threshold levels for an alert

To create an agent and subscribe the agent to the `ALERT_QUE`, complete the following steps:

1. Run the `CREATE_AQ_AGENT` procedure of the `DBMS_AQADM` package.
2. Run the `ADD_SUBSCRIBER` procedure of the `DBMS_AQADM` package.
3. Associate a database user with the subscribing agent, because only a user associated with the subscribing agent can access queued messages in the secure `ALERT_QUE`.
4. Assign the enqueue privilege to the user by running the `ENABLE_DB_ACCESS` and `GRANT_QUEUE_PRIVILEGE` procedures of the `DBMS_AQADM` package.
5. Register with the `DBMS_AQ.REGISTER` procedure to receive an asynchronous notification when an alert is enqueued to `ALERT_QUE`. The notification can be in the form of email, HTTP post, or PL/SQL procedure.

To read an alert message, complete the following steps:

1. Use the `DBMS_AQ.DEQUEUE` procedure or `OCIAQDeq` call.
2. After the message has been dequeued, use the `DBMS_SERVER_ALERT.EXPAND_MESSAGE` procedure to expand the text of the message.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_AQ` package
- *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_AQADM` package

6.1.2.4 Server-Generated Alerts Data Dictionary Views

You can query data dictionary views for information about server-generated alerts.

View	Description
DBA_THRESHOLDS	Lists the threshold settings defined for the instance
DBA_OUTSTANDING_ALERTS	Describes the outstanding alerts in the database
DBA_ALERT_HISTORY	Lists a history of alerts that have been cleared
V$ALERT_TYPES	Provides information such as group and type for each alert
V$METRICNAME	Contains the names, identifiers, and other information about the system metrics
V$METRIC	Contains system-level metric values
V$METRIC_HISTORY	Contains a history of system-level metric values

6.2 Monitoring Performance

Monitoring performance includes monitoring locks and wait events and querying a set of data dictionary views.

Monitoring database performance is covered in detail in *Oracle Database Performance Tuning Guide* and *Oracle Database SQL Tuning Guide*.

- Monitoring Locks
  Locks are mechanisms that prevent destructive interaction between transactions accessing the same resource. The resources can be either user objects, such as tables and rows, or system objects not visible to users, such as shared data structures in memory and data dictionary rows.
- About Monitoring Wait Events
  Wait events are statistics that are incremented by a server process to indicate that it had to wait for an event to complete before being able to continue processing. A session could wait for a variety of reasons, including waiting for more input, waiting for the operating system to complete a service such as a disk write, or it could wait for a lock or latch.
- Performance Monitoring Data Dictionary Views
  You can query a set of data dictionary views to monitor an Oracle Database instance.

6.2.1 Monitoring Locks

Locks are mechanisms that prevent destructive interaction between transactions accessing the same resource. The resources can be either user objects, such as tables and rows, or system objects not visible to users, such as shared data structures in memory and data dictionary rows.

Oracle Database automatically obtains and manages necessary locks when executing SQL statements, so you need not be concerned with such details. However, the database also lets you lock data manually.

A deadlock can occur when two or more users are waiting for data locked by each other. Deadlocks prevent some transactions from continuing to work. Oracle Database automatically detects deadlock situations and resolves them by rolling back one of the statements involved in the deadlock, thereby releasing one set of the conflicting row locks.

Oracle Database is designed to avoid deadlocks, and they are not common. Most often they occur when transactions explicitly override the default locking of the database. Deadlocks can affect the performance of your database, so Oracle provides some scripts and views that enable you to monitor locks.

To monitor locks:

1. Run the `catblock.sql`, which creates lock views.
2. Run the `utllockt.sql` script, which uses the views created by `catblock.sql` to display, in a tree fashion, the sessions in the system that are waiting for locks and the locks that they are waiting for.

The location of the script files is operating system dependent.

> **See Also:**
>
> - "Performance Monitoring Data Dictionary Views"
> - *Oracle Database Concepts* contains more information about locks.

6.2.2 About Monitoring Wait Events

Wait events are statistics that are incremented by a server process to indicate that it had to wait for an event to complete before being able to continue processing. A session could wait for a variety of reasons, including waiting for more input, waiting for the operating system to complete a service such as a disk write, or it could wait for a lock or latch.

When a session is waiting for resources, it is not doing any useful work. A large number of waits is a source of concern. Wait event data reveals various symptoms of problems that might be affecting performance, such as latch contention, buffer contention, and I/O contention.

Oracle provides several views that display wait event statistics. A discussion of these views and their role in instance tuning is contained in *Oracle Database Performance Tuning Guide*.

6.2.3 Performance Monitoring Data Dictionary Views

You can query a set of data dictionary views to monitor an Oracle Database instance.

These views are general in their scope. Other views, more specific to a process, are discussed in the section of this book where the process is described.

View	Description
`V$LOCK`	Lists the locks currently held by Oracle Database and outstanding requests for a lock or latch
`DBA_BLOCKERS`	Displays a session if it is holding a lock on an object for which another session is waiting
`DBA_WAITERS`	Displays a session if it is waiting for a locked object
`DBA_DDL_LOCKS`	Lists all DDL locks held in the database and all outstanding requests for a DDL lock
`DBA_DML_LOCKS`	Lists all DML locks held in the database and all outstanding requests for a DML lock

View	Description
DBA_LOCK	Lists all locks or latches held in the database and all outstanding requests for a lock or latch
DBA_LOCK_INTERNAL	Displays a row for each lock or latch that is being held, and one row for each outstanding request for a lock or latch
V$LOCKED_OBJECT	Lists all locks acquired by every transaction on the system
V$SESSION_WAIT	Lists the resources or events for which active sessions are waiting
V$SYSSTAT	Contains session statistics
V$RESOURCE_LIMIT	Provides information about current and maximum global resource utilization for some system resources
V$SQLAREA	Contains statistics about shared SQL area and contains one row for each SQL string. Also provides statistics about SQL statements that are in memory, parsed, and ready for execution
V$LATCH	Contains statistics for nonparent latches and summary statistics for parent latches

6.3 Monitoring Quarantined Objects

Object quarantine enables an Oracle database to function even when there are corrupted, unrecoverable objects. The V$QUARANTINE view contains information about quarantined objects.

- About Object Quarantine
  Object quarantine isolates an object that has raised an error and monitors the object for impacts on the system.
- Viewing Quarantined Objects
  The V$QUARANTINE view stores information about the objects that are currently quarantined.

6.3.1 About Object Quarantine

Object quarantine isolates an object that has raised an error and monitors the object for impacts on the system.

Some Oracle Database errors, such as ORA-00600 and ORA-07445, typically cause the process to terminate, which can cause the database to terminate. When such an error is encountered, object quarantine attempts to isolate the resource that caused the error so that the database can continue to run. The resource is isolated in memory so that it does not affect the rest of the database. The V$QUARANTINE view stores information about the objects that are currently quarantined.

Most database resources can raise errors that can cause a database to terminate. For example, library cache memory objects can raise such errors.

In a multitenant environment, a multitenant container database (CDB) can, in some cases, use object quarantine to isolate and terminate a pluggable database (PDB) that has raised a serious error instead of terminating the CDB.

A quarantined resource typically remains quarantined until the database is restarted. If a resource is quarantined for a PDB in a CDB, then the resource is quarantined until the PDB is closed and re-opened.

6.3.2 Viewing Quarantined Objects

The `V$QUARANTINE` view stores information about the objects that are currently quarantined.

1. Connect to the database as an administrative user.
2. Query the `V$QUARANTINE` view.

Example 6-1 Querying the V$QUARANTINE View

This query shows the resources that are currently quarantined.

```
COLUMN OBJECT FORMAT A10
COLUMN ADDRESS FORMAT A10
COLUMN BYTES FORMAT 999999999
COLUMN ERROR FORMAT A20
COLUMN TIMESTAMP FORMAT A20

SELECT OBJECT, ADDRESS, BYTES, ERROR, TIMESTAMP
   FROM V$QUARANTINE;
```

Your output is similar to the following:

```
OBJECT     ADDRESS         BYTES ERROR                TIMESTAMP
---------- ---------- ---------- -------------------- --------------------
session    0000000078       9528 ORA-00600: internal  16-SEP-15 01.17.42.2
           B54BC8                error code, argument 85878 PM -07:00
                                 s: [12345], [], [],
                                 [], [], [], [], [],
                                 [], [], [], []
```

This output shows the following about the quarantined resource:

- The name of the resource is “session.”
- The start address of the memory region being quarantined is 0000000078B54BC8. Typically, this is the address of the resource, such as the session in this example.
- The resource is using 9528 bytes of memory in quarantine.
- The message of the error that caused the resource to be placed in quarantine is “ORA-00600 internal error code.”
- The timestamp shows the date and time of the error.

6.4 Automatically Monitoring Schema Objects

Oracle Database can automatically track the activities and usage of certain schema objects, such as tables and materialized views.

The Object Activity Tracking System (OATS) tracks various activities associated with database objects. Tracking can be performed both at the database level and pluggable database (PDB) level. The activities tracked include DML operations on tables, table and partition scans, partition maintenance operations (PMOPs), materialized view rewrite and refresh, and usage of auxiliary structures such as indexes. The statistics are used to support automated database functionality such as automatic materialized views.

Use procedures and functions in the `DBMS_ACTIVITY` package to control the information captured by OATS.

To enable Object Activity Tracking System:

- Set the `STATISTICS_LEVEL` initialization parameter to `TYPICAL` or `ALL`.

Statistics tracked by OATS can be viewed in the following data dictionary views: `DBA_ACTIVITY_CONFIG`, `DBA_ACTIVITY_SNAPSHOT_META`, `DBA_ACTIVITY_TABLE`, and `DBA_ACTIVITY_MVIEW`.

Related Topics

- *Oracle Database Data Warehousing Guide*