

An Oracle Technical White Paper  
April 2009

# The Self-Managing Database: Deploying Oracle Database 11g in Embedded Environments

Introduction .....	1
Why Embed?.....	2
ISV/OEM Benefits .....	2
Customer Benefits.....	2
Challenges of an Embedded Database .....	2
Deploying an Embedded Database .....	3
Oracle Database 11g Installation Enhancements .....	3
Instant Client .....	5
Designing a Self-Managing Database.....	5
Automation of Routine Administrative Tasks .....	5
Configuring Adaptive Systems .....	12
Designing Self-Reliant Systems.....	15
Diagnosing and Resolving Problems .....	20
Automatic Database Diagnostic Monitor .....	20
Conclusion .....	22

## Introduction

IT departments of all sizes are increasingly relying on self-managing systems. The key benefit, of course, is cost savings—but self-managing systems also address organizations' lack of onsite technical expertise and the nonfeasibility of managing systems individually.

Solution providers often seek to provide systems with embedded components that require little to no administration. Embedded installations enable ISVs and OEMs to build critical technology directly into their products so they can develop solutions targeted specifically to the needs of their market. In an embedded solution, database software is fully integrated into the business application or device so the end user has little or no knowledge that the database software exists. In essence, the database is completely invisible to the application user. This gives ISVs and OEMs greater control of the complete application—from installation to management and support.

With Oracle Database 11g, Oracle has made large strides toward creating a truly embeddable database. This white paper reviews the technical challenges associated with embedding a database and how Oracle Database 11g addresses them.

## Why Embed?

Embedded databases have many benefits—not only for ISVs and OEMs developing a business application, but also for their end users or customers.

### ISV/OEM Benefits

ISVs and OEMs that embed a database into their application can

- Reduce the total cost of ownership of the solution, thereby offering cost advantages to their customers.
- Provide a single integrated solution that requires minimal management.
- Test the database configuration that best suits their application and use the same configuration to configure customers' databases.
- Manage and upgrade customers' databases more easily and at lower cost. Because the embedded database is installed, configured, and managed by the ISV/OEM application, its installation and configuration is easy, fast, and less error-prone, thus significantly reducing support costs.

### Customer Benefits

Most of the benefits mentioned above for ISVs and OEMs are directly transferable to customers. With an embedded solution, customers can

- Reduce support issues by working with one vendor and, conceptually, one product
- Reduce IT costs by avoiding the need for DBAs or specialists to manage their databases
- Increase the speed of application installation by installing single-application software rather than separate installations of database and application software
- Enjoy high performance of the application as a result of a well-configured database environment

These are just some of the benefits that ISVs, OEMs, and end customers can receive by embedding and using Oracle Database 11g in their applications.

## Challenges of an Embedded Database

An embedded database must address certain challenges unique to the environment. These challenges can be classified into the following categories:

- **Deployment.** Installation and configuration of an embedded database must be completely invisible to the users and fully integrated with their application. The database must be packaged as a component of the application package and must be installed as part of the application installation.
- **Management.** An embedded database must be self-managing, with all routine administrative tasks automated. In addition, the database must be able to adapt to changes in system environment and should be able to address common problems automatically, without requiring outside intervention. In the rare event of errors or problems, the database should provide an easy way to diagnose the problem and suggest possible remedies.
- **Software maintenance and support.** Software maintenance is a necessary aspect of all software deployments. For embedded databases, it is essential that tasks such as upgrades and software patching be made very simple and as automated as possible.

The remainder of this paper discusses each of these challenges in turn.

## Deploying an Embedded Database

*This section discusses embedded software deployment at a high level and suggests further reading for additional information.*

Oracle Database 11g offers a variety of enhancements that aid in the database-embedding process.

### Oracle Database 11g Installation Enhancements

Oracle Database has significantly simplified installation and configuration since the release of Oracle Database 10g. Oracle Database 11g installation can be run in a true silent mode for installing the software. Along with silent installation, Oracle Database 11g supports silent mode deinstallation. Any program can call the Oracle installer in silent mode and provide the required parameters via the command line. The new approach to deploying an embedded database uses the cloning capabilities of the installer.

Oracle Database 11g installation also configures the Oracle software, a listener, and a database all in a single integrated process. You can replace the Oracle-provided precreated seed database with your own application-specific database. This capability enables you to speed up application installation. You can create and configure a database, load it with its application-specific metadata in your own lab, and then package and ship it on media as an integrated component of your application. For those who would like to have more-customized database creation, such as a specific datafile location, Oracle Database also provides flexibility for creating and configuring a database based on command-line arguments.

Embedding an Oracle Database instance can be divided into three phases:

- **Preparation.** A source database system is prepared for embedding into the integrated solution that will be delivered to the customer.
- **Packaging.** Software binaries and data are packaged with the application bundle.
- **Installation.** The customer uses the package to install the application with the embedded database on the customer machines.

In most cases, this is a much more efficient approach to installing Oracle software than the traditional approach of silent install. Cloning requires a source system that matches the target system's processor architecture and operating system.

### Preparation

The first stage in the invisible deployment process is the preparation of a preinstalled Oracle Database instance for cloning and deployment in an embedded environment. The preparation phase includes creating database objects such as users, tables, and indexes; applying relevant patches to Oracle software; and creating metric thresholds and any necessary corrective action scripts.

### Packaging

This step primarily involves the bundling of Oracle software binaries and the database template with your application installation processes. Any installation software should be able to interface with the scripts required for cloning the Oracle Database instance. Packaging involves creating a template out of the data by use of the Oracle Database Configuration Assistant tool. The template includes the metadata and may also include the datafiles. The template files can be included in the same directory structure as the Oracle software or be packaged separately with installation software. Once the template is ready, the next step is to create an archive of the Oracle software binaries, which can be compressed with any of the widely available compression tools, such as `gzip` on Linux.

### Installation

This step is carried out on the target system. It involves the extraction of Oracle binaries from the archive included in the application distribution, followed by silent network configuration and, finally, by the silent creation of the database—from the included template, for example.

The installation of Oracle software is customizable to some extent. For example, the Oracle base and home directories as well as the home name can be different from the source.

## Instant Client

Another important enhancement for applications connecting to an Oracle Database instance is that it is no longer necessary to install an Oracle client (drivers only) into an Oracle home directory. An application can package Oracle drivers, copy them anywhere, and then simply set the necessary environment variables on UNIX and/or registry entries on the Windows platform.

The Oracle Database Instant Client libraries are available as a download from Oracle Technology Network (OTN) for distribution with your solution. Further details are available here:

[oracle.com/technology/tech/oci/instantclient/index.html](http://oracle.com/technology/tech/oci/instantclient/index.html)

Further details of the embedding process can be found in the Oracle Embedded Installation Resource Kit 11g here:

[oracle.com/technology/tech/embedded/files/oeikit11g.zip](http://oracle.com/technology/tech/embedded/files/oeikit11g.zip)

## Designing a Self-Managing Database

The second challenge an embedded database must meet is that it must be self-managing—no onsite maintenance should be required for day-to-day operations. This is a requirement because embedded environments typically do not have access to onsite DBAs or other technical expertise that can actively manage the database systems. In most cases, users have some technical knowledge of the specific application they are using, but not of the database system that runs underneath the application.

For such environments, the database instance must be self-managing once it is deployed, with all routine administrative tasks fully automated. It should also be able to adapt to changes in the environment without requiring outside intervention. And when problems do occur, it should be able to repair the problem single-handedly, again without requiring any outside intervention.

### Automation of Routine Administrative Tasks

Embedded databases should include automated backup-and-recovery management, space management, and performance management.

#### Backup-and-Recovery Management

Performing regular backups to recover from data loss is perhaps the most critical administrative task, and its automation is essential for an embedded database. Oracle recommends that a backup with the following characteristics be created.

- **Uses Oracle Recovery Manager (Oracle RMAN) for performing any backup-and-recovery operations.** Oracle RMAN is Oracle's utility for managing the backup and, more

importantly, the recovery of the database. It eliminates operational complexity while providing superior database performance and availability. Oracle RMAN determines the most efficient method of executing the requested backup, restore, or recovery operation and then executes the operation in concert with the Oracle Database server. It provides capabilities such as incremental backup and corrupt block detection while taking care of backup, block media recovery, and other tasks to ease the backup-and-recovery administration of an Oracle Database instance.

- **Performs on-disk backups by using the flash recovery area feature.** The flash recovery area is a unified storage location for all recovery-related files and activities in an Oracle Database instance. By defining one initialization parameter, all Oracle RMAN backups, archive logs, control file autobackups, and datafile copies are automatically written to a specified disk location. The flash recovery area is completely self-managing. The Oracle Database server automatically ages out old backup and archive log files and maintains only the current ones. Users no longer need to worry about keeping the flash recovery area size small by deleting old backups, nor do they need to keep track of the archive logs needed for recovery. This speeds up the recovery process, because all files needed for any recovery activity are located in one place and Oracle Database is aware of that location. Backup-and-recovery activity in the flash recovery area is very fast, because disk access is much faster and more efficient than tape access. On current disks, data anywhere on the disk can be accessed in a few milliseconds.

You can define the flash recovery area by setting the following two initialization parameters:

- `DB_RECOVERY_FILE_DEST` sets the location of the flash recovery area
- `DB_RECOVERY_FILE_DEST_SIZE` sets the size in bytes of the flash recovery area

Both of these parameters are dynamic and can be altered or disabled by use of the `ALTER SYSTEM SET` command with the database online.

- **Performs full database backup once, and then performs incremental nightly backups.** Incremental backups in Oracle Database are very efficient: because only the changed blocks are backed up, the datafiles do not have to be scanned to identify the changed blocks. Oracle Database 11g keeps track of the changed blocks in each datafile in a special change-tracking file (for a 1TB database, the approximate size of this file is 4MB), so that when incremental backups are performed, Oracle Database automatically uses the change-tracking file and backs up only the changed blocks, without the overhead of datafile scanning. This makes incremental backups fast and space-efficient. By default, Oracle Database does not keep track of changed blocks—so you should enable this feature in order to benefit from incremental backup enhancements. The command for enabling block change tracking is

```
ALTER DATABASE ENABLE BLOCK CHANGE TRACKING;
```

Another enhancement in Oracle Database is the ability to merge an incremental backup with the image copy of the database. This extremely powerful feature rolls the image backup copy forward, thereby reducing the time needed for media recovery; it also saves space, because the

incremental backup that has been merged can now be deleted. This means that a full database backup must be performed only once in the lifetime of a database, and the size of the flash recovery area will stabilize and not grow much beyond the actual database size. On a weekly basis, incremental backups should be merged into the image copy of the database backup. The command for merging the incremental backup into the image copy by rolling it forward is

```
RECOVER COPY OF DATAFILE <datafile# or name>
```

Starting with Oracle Database 10g, when a database instance is created with Oracle Database Configuration Assistant, a backup job is automatically created that performs nightly incremental backups to the flash recovery area. By default, this job is created as an Oracle Enterprise Manager scheduler job. Users can use this predefined job or, if they like, they can define the same job manually by using the server-based unified scheduler. Note that for the unified scheduler, users must create the backup script themselves with the previously recommended settings.

### Space Management

While backup management is the most critical of routine administrative functions, space management is perhaps the most common. In an environment with no DBA, a database must be configured to minimize, if not eliminate, any space-management-related tasks. Starting with enhancements from as far back as Oracle8i, Oracle has steadily automated more and more space management functions, culminating in Oracle Database 10g, where the solution was finally complete. Five critical space-management solutions should be utilized in an embedded database, as described in the next sections.

### Oracle-Managed Files

The Oracle-managed files (OMF) feature was introduced in Oracle9i Database. It eliminates the need to manually administer database files by specifying a default location for datafiles, control files, and online log files. When adding or creating a new file, the user does not need to specify its location or size. Oracle Database automatically generates a unique name for the file, creates it, and deletes it from the OS when the corresponding object is dropped from the database. To enable this feature, specify the following initialization parameters:

- `DB_CREATE_FILE_DEST` specifies the default location of datafiles
- `DB_CREATE_ONLINE_LOG_DEST_<n>` specifies the default location for copies of redo logs and control files

OMF datafiles, created by default, are 100MB in size and are autoextensible, with an unlimited maximum size. The default size of OMF online logs is also 100MB.

OMF has many benefits. It facilitates creation and deletion of database files by eliminating the need to put OS-specific filenames and sizes into scripts. It also reduces the possibility of human error (for example, specifying incorrect filenames) and reduces wasted disk space taken up by

obsolete files. All these factors make it very suitable for automation of many tasks related to file creation and management, such as adding a datafile when a tablespace encounters an out-of-space condition.

### **Automatic Undo Management**

The automatic undo management (AUM) feature was introduced in Oracle9 Database. It enables the database server to self-manage undo segments, freeing the user from creating and managing rollback segments. Users no longer need to assign certain transactions to specific rollback segments, nor do they need to determine the right size and number of rollback segments. The only prerequisite is the creation of an undo tablespace—the database will do the rest.

AUM has many additional benefits. It essentially eliminates the possibility of undo block and consistent read contention, because the server dynamically adjusts the space allocated to the undo segments to meet the current workload requirement. Furthermore, by sizing the undo tablespace properly, users can completely avoid the dreaded ORA-1555 (“snapshot too old”) error.

The only configuration decision that must be made is regarding the size of the undo tablespace. Oracle Database 11g provides an undo advisor which guides in sizing the tablespace. However, for embedded databases, it is advisable to make the undo tablespace autoextensible. This tablespace attribute should be set to YES, which will enable the undo tablespace to grow automatically as needed, based on application workload.

### **Locally Managed Tablespaces**

Locally managed tablespaces were introduced in Oracle8 Database. They perform better than dictionary-managed tablespaces, are easier to manage, and eliminate space-fragmentation-related problems. For operations such as space allocation and deallocation, they use bitmaps stored in datafile headers and, unlike dictionary-managed tablespaces, do not contend for centrally managed resources. They eliminate the need for many recursive operations that are sometimes required for dictionary-managed space allocation, such as allocating a new extent.

Locally managed tablespaces offer two options for extent management: Auto Allocate and Uniform. In the Auto Allocate option, the Oracle Database server itself determines the size of each extent allocated, while in the Uniform option, all extents in that tablespace are of the size specified by the UNIFORM clause of the CREATE TABLESPACE statement. It’s best to use the Auto Allocate option, even though it may result in objects with multiple extents, because locally managed tablespaces can handle numerous extents (more than 1,000 per object) without any noticeable performance impact. Because there’s no need to worry about extent sizes, Oracle recommends use of the Auto Allocate option.

Locally managed tablespaces address two important issues that are critical for embedded environments. First, they eliminate external fragmentation of a tablespace; second, they handle

extent sizing and management internally, so you don't need to defragment tablespaces to recover lost space or enhance performance.

### **Automatic Segment Space Management**

Automatic segment space management (ASSM) was introduced in Oracle9 Database to automatically handle space management within a segment. Prior to Oracle9i Database, segment attributes such as FREELISTS, FREELIST GROUPS, and PCTUSED had to be configured manually. As a database environment changes over time due to factors such as data growth or an increase in concurrency, these attributes may have to be reconfigured manually by re-creation of the segment. With ASSM, no manual configuration or reconfiguration is needed.

ASSM makes space management within a segment completely transparent by using bitmaps instead of FREELISTS for managing space utilization within each data block. ASSM also improves the space utilization of a data block, because the bitmaps are much better equipped to track and manage free space at the data-block level than FREELISTS. ASSM enables better reuse of available free space, particularly for segments with rows of varying size, and improves the performance of concurrent data manipulation language (DML) operations significantly, because different parts of the bitmap can be used simultaneously, eliminating serialization for free space lookups.

### **Online Segment Shrink**

Online segment shrink is a very useful Oracle Database feature for embedded databases. Segments that undergo frequent insert-and-delete DML activity can become internally fragmented and develop unusable free space in the data blocks. This results in wasted space in the segment, which takes up much more space than is really needed. Moreover, this wasted space within a data block has a negative performance impact because it causes unnecessary row chaining. Online segment shrink reclaims such wasted space in segments by shrinking them.

Segment shrink involves moving row pieces around within a segment to fill up the wasted space. It also removes row chaining wherever possible, thus improving the performance of read operations on the segment. The shrink operation is online and in place, meaning that activity on the segment can continue when the operation is being performed. It does not need additional space for storing temporary data, in contrast to other online operations such as online data redefinition and reorganization.

Oracle Database 11g provides a segment advisor that recommends—based on the amount of wasted space—which segments are good candidates for shrinking. It takes a tablespace name as an input and makes recommendations for all the segments within that tablespace. A job should be created by use of the unified scheduler that runs weekly—calling the segment advisor on all the application tablespaces that undergo DML activity—and implements the advisor recommendations. This will ensure more-efficient space utilization within a segment and will also reduce the possibility of encountering out-of-space conditions.

The following SQL\*Plus script gives an example of how the segment advisor can be called by use of a PL/SQL interface and its recommendations implemented automatically. This script runs the segment advisor on the tablespace USERS and then shrinks all of its segments that are good candidates for shrinking.

```
variable id number;
begin
declare
    name varchar2(100) ; descr varchar2(500) ; objid number;
begin
    name := ' ' ;
    descr := 'Segment Advisor Demo';
    dbms_advisor.create_task('Segment Advisor', :id, name, descr, NULL) ;
    dbms_advisor.create_object(name, 'TABLESPACE', 'USERS', NULL, NULL,
NULL, objid);
    dbms_advisor.set_task_parameter(name, 'RECOMMEND_ALL', 'TRUE') ;
    dbms_advisor.execute_task(name) ;
end ;
end ;
/

set echo off
spool shrink.sql
select attr1 || ';' from dba_advisor_actions where task_id=:id ;
spool off
@shrink
```

Locally managed tablespaces handle the tablespace space fragmentation—or external fragmentation—problem, and now creating a regular job that performs online segment shrink also eliminates the problem of segment space fragmentation, or internal fragmentation. Thus, with both locally managed tablespaces and online segment shrink, all space-related fragmentation issues are now comprehensively addressed.

Together, the previously mentioned solutions provide a comprehensive way to handle the automation of all space management functions. In embedded database environments, they are a necessary prerequisite for eliminating the need for onsite database maintenance.

### Performance Management: Optimizer Statistics Collection

Application performance is entirely dependent upon the ability of the Oracle cost-based optimizer (CBO) to produce optimal execution plans. The CBO relies on the availability of valid statistics on objects and their data, known as optimizer statistics, to generate execution plans that are fast and efficient. Invalid or stale optimizer statistics would lead to the optimizer producing suboptimal or even bad execution plans, resulting in poor application performance.

Over time, as objects undergo DML activity, optimizer statistics can get unrepresentative or stale, meaning that optimizer statistics have to be refreshed regularly to ensure that the CBO always

has valid statistics available. Doing this task efficiently, especially when large volumes of data are involved, can be nontrivial. You need to keep track of objects whose data has undergone significant change to warrant re-collection of statistics, determine appropriate sample size for statistics collection, ascertain which table columns require histograms, determine the appropriate degree of parallelism, and so on, before statistics can be collected. In an embedded database environment where there are no DBAs to perform this very essential routine function, a job must be scheduled that runs nightly and efficiently collects and/or refreshes the necessary optimizer statistics.

In Oracle Database 11g, this task is completely automated. A job is created by default that refreshes statistics on objects with stale or no statistics. It does the following:

- Uses the modification-monitoring feature to track which objects have been modified, and to what degree, to identify candidates for statistics collection.
- Analyzes objects in the order of the extent of their modification. Objects with the most rows modified will be analyzed first, followed by the next-most-modified, and so on.
- Automatically determines
  - The sample size for statistics collection
  - Which table columns require histograms
  - The degree of parallelism for the statistics collection job
  - Granularity for partitioned tables, such as whether to collect statistics at the global, partition, or subpartition level
- Gathers statistics on dictionary objects. This is very important, because—starting with Oracle Database 10g—data dictionary statements use the CBO.

No special configuration is needed in Oracle Database 11g to turn on automatic collection of optimizer statistics. The job is scheduled by the unified scheduler to run nightly, out of the box, in a predefined maintenance window. The parameters of this window—time, frequency, and so on—are all modifiable. Moreover, with the resource manager, the CPU consumption in the maintenance window can also be managed so that its jobs do not consume excessive CPU resources when some important batch job or transaction may be running on the system.

One cautionary note: Like any other unified scheduler job, the optimizer statistics collection job can be disabled, so make sure that the job is always enabled. It is an essential requirement for embedded database installations.

## Configuring Adaptive Systems

An embedded database should be configured so that it can adapt to basic fluctuations in system workload. Otherwise, system performance can become unstable and the possibility of encountering exception conditions such as ORA-4031 (“out of shared memory”) or ORA-1555 (“snapshot too old”) can increase. This section discusses configuration of several key areas that enable the database to be more adaptive so that it can seamlessly handle workload variations.

### Autopartition Management

Interval partitioning is a new manageability feature of Oracle Database 11g that automates the creation of range partitions based on specified intervals. This is a very useful feature in applications that need to maintain partitions for out-of-range data. New partitions are created only as needed, thus freeing up valuable disk space. Moreover, the application is spared the overhead of creating new partitions for data that may not arrive until much later.

Interval partitioning can also be used for the following composite partitioning strategies: Interval-range, Interval-hash, and Interval list.

```
CREATE TABLE sales_interval_part
(order_date DATE, a number, b varchar2(22))
PARTITION BY RANGE (order_date)
INTERVAL(NUMTOYMINTERVAL(1, 'month'))
(PARTITION P0 values less than
   (TO_DATE('1-FEB-2000', 'dd-MON-yyyy')));
```

### Automatic SQL Tuning

Because the workload of an application can change continuously and require several manual steps to tune high-load SQL, automation of the tuning process is critical in an embedded environment. SQL tuning is a difficult task in embedded databases, but a new Oracle Database 11g feature makes this task easier by completely automating the tuning procedure.

Oracle Database detects high-load SQL statements and then uses the maintenance window to tune them. SQL profiles are created out of the high-load statements chosen from the workload. The SQL tuning advisor collects auxiliary statistics about the SQL, using sampling and partial execution techniques to build the SQL profile with better estimates of the statistics required to create a more optimal plan.

The automatic SQL tuning task runs in the nightly maintenance window. Candidate SQL (statements with significant impact in terms of CPU time and I/O) is identified from the automatic workload repository (AWR). These statements are tuned individually by the SQL tuning advisor, which creates a SQL profile for the statement and tests it with and without the

profile. If the performance difference is at least a factor of 3, the SQL profile is automatically implemented.

SQL profiles are different from stored outlines because they do not capture the plan and freeze it. With different indexes or data growth, the plan can change and still have a relevant SQL profile.

### **Automatic Memory Tuning**

Two new features in Oracle Database 11g offer improved memory management capabilities.

#### **Automatic Memory Management**

Oracle Database 11g takes automatic memory tuning a step further with the automatic memory management feature, which enables Oracle Database to automatically redistribute instance memory between the SGA and PGA memory structures, based on the workload in real time. By using this feature, applications can benefit from better memory distribution and avoid performance issues due to incorrectly sized parameters.

The new parameters in Oracle Database 11g are

- MEMORY\_TARGET
- MEMORY\_MAX\_TARGET

The automatic SQL execution memory feature was introduced in Oracle9i Database. This feature enables the database server to automatically distribute SQL execution memory among various active processes in an intelligent manner to ensure maximum performance benefits and the most efficient utilization of memory. This means that when the workload on a system changes, the server will automatically adapt to it by reallocating memory among various processes as needed.

This behavior is very different from the old behavior, when SQL execution memory allocation was static. Initialization parameters such as SORT\_AREA\_SIZE, HASH\_AREA\_SIZE, BITMAP\_MERGE\_AREA\_SIZE, and CREATE\_BITMAP\_AREA\_SIZE were specified to control the maximum amount of memory allocated to SQL work areas for performing operations such as a sort or hash join. The memory allocated to each process was static and could not be redistributed to other SQL work areas as the workload changed. This resulted in poor performance for systems with varying workloads, because in this type of SQL execution memory management, the system was really configured for a single workload. Systems with onsite DBAs worked around this problem by having different configuration settings for different workloads. As the workload changed, such as from online transaction processing (OLTP) to batch, they would modify the initialization parameter settings accordingly.

In an embedded environment without an onsite DBA, this is not possible. Hence, using the automatic SQL execution memory management feature is essential for such environments. Enabling this feature involves simply setting one initialization parameter,

PGA\_AGGREGATE\_TARGET, to the maximum amount of PGA memory available to the database. The Oracle Database server will then take over and manage the SQL execution memory automatically.

#### **Automatic Shared Memory Tuning**

Automatic shared memory tuning solves exactly the same problem for shared memory that automatic SQL execution memory tuning solves for private or SQL execution memory. It allows for dynamic redistribution of shared memory between various memory pools—such as buffer cache, shared pool, large pool, and Java pool—based on workload changes.

For example, a batch workload may need a bigger large pool for parallel queries, whereas an OLTP workload might perform better with a larger buffer cache. With automatic shared memory tuning, Oracle will adjust the size of the memory pools by allocating a bigger large pool when a batch workload is running; when the workload switches to OLTP, it will deallocate memory from the large pool and allocate it to buffer cache to get optimal OLTP performance without any external intervention.

Prior to Oracle Database 10g, each shared memory pool had to be tuned for different workloads separately. In non-DBA environments, this meant that the system could, at best, be tuned for a single workload and when workloads changed, the database would perform below par and possibly even encounter exception conditions such as ORA-4031 (“unable to allocate xxx bytes of shared memory”). Automatic shared memory tuning not only adapts to different workloads dynamically and without outside intervention, but it also uses shared memory more efficiently and enhances the overall performance of the system. This feature is essential for embedded environments. Enabling it is a matter of simply setting the initialization parameter SGA\_TARGET to the total amount of shared memory available to the database instance.

#### **Automatic Undo Retention Tuning**

Automatic undo retention tuning is an Oracle Database feature available only when automatic undo management (AUM) is enabled. This feature dynamically adjusts undo retention for a given size of the undo tablespace and the workload on the system.

In manual undo retention tuning, a user would manually set the initialization parameter UNDO\_RETENTION to the length of time the longest-running query is expected to take. Even if this parameter is initially set correctly, the time taken by the longest-running query on the system may change as the data distribution changes over time. If this time increases, users can encounter an ORA-1555 (“snapshot too old”) error, and if the time decreases, the space in the undo tablespace will be underutilized. The former scenario—an increase in the time taken by the longest query—is more typical and its consequences more serious. It results in exceptions that cause queries and/or transactions to fail.

The only way to fix the problem in manual undo retention mode is by manually changing the UNDO\_RETENTION parameter. Once again, for embedded databases, this is not acceptable, so embedded databases should be configured to use automatic undo retention tuning, which dynamically changes undo retention to keep it ahead of the longest query on the system. With this feature enabled and an autoextensible undo tablespace (recommended earlier), the likelihood of encountering errors such as ORA-1555 is almost nil.

## Designing Self-Reliant Systems

Automating routine administrative tasks and configuring the database to be adaptive fulfill a major requirement for systems with no onsite DBAs. However, even in a perfect setup, errors or exceptions will eventually occur. It is therefore imperative that the database be designed to handle such exceptions gracefully and be able to recover and self-correct them. Oracle Database provides several solutions to facilitate the creation of such self-reliant systems.

A self-reliant system must do two things. First, it must be able to trap or identify any exception, and second, it should be able to take corrective action to remedy the problem. Oracle Database 11g provides server-generated alerts for some of the most common types of exceptions. These include space alerts when a tablespace is running out of space, ORA-1555 alerts, alerts associated with space problems in the flash recovery area, and a host of performance metrics alerts. This takes care of the first requirement for a self-reliant system: the ability to trap or identify exception conditions.

The second requirement, providing the ability to take corrective actions, is also provided by Oracle Database. Two options are available here. The simpler one involves using Oracle Enterprise Manager's "fix-it" job capability. In Oracle Enterprise Manager, for each kind of alert generated, a response action can be defined in the form of an operating system command or a script. This basic framework can be used to set up a self-reliant embedded database. The second option, using the Oracle Advanced Queuing callback capability, is possible because Oracle Database 11g's server-generated alerts use the Oracle Advanced Queuing infrastructure.

In Oracle Database 11g, a queue called ALERT\_QUE is created by default and is used for all alerts generated by the server-generated-alerts feature. A user can subscribe to this queue to catch a particular exception—such as ORA-1555 or a tablespace-full condition—or recovery area alert and then register a user-defined callback procedure (a PL/SQL, Java Message Service (JMS), or Oracle Call Interface (OCI) function) to it, so that every time the particular exception occurs, the callback procedure is triggered. This callback procedure will contain the user-defined remedy for the exception.

For example, for an ORA-1555 exception, the callback procedure remedy could include something like calling the undo advisor to determine the right size of the undo tablespace and then adjusting its size accordingly. As a result, the next time the user runs the same transaction or query that resulted in ORA-1555, the error will not occur, because the undo tablespace would

have been adjusted automatically. The following shows in a step-by-step fashion how Oracle Advanced Queuing can be used for creating self-reliant systems.

1. Create three new subscribers to ALERT\_QUEUE to catch ORA-1555, tablespace out-of-space condition, and recovery area alerts. Example code for the three subscribing agents is shown below. Note that the only difference between the three subscriptions is the subscriber name and its rule.

```

DECLARE
  subscriber      aq$_agent;
BEGIN
  subscriber := aq$_agent('UNDO_SUB','ALERT_QUEUE',null);
  dbms_aqadm.add_subscriber(
    queue_name     => 'ALERT_QUEUE',
    subscriber     => subscriber,
    rule           => 'tab.user_data.reason_id = 10 or
tab.user_data.reason_id =11' );
END;
/

```

**Example 1: ORA-1555 error subscription**

```

DECLARE
  subscriber      aq$_agent;
BEGIN
  subscriber := aq$_agent('SPACE_SUB','ALERT_QUEUE',null);

-- shows how to catch space exception for specific tablespace (TEST_TBS)
dbms_aqadm.add_subscriber(
  queue_name      => 'ALERT_QUEUE',
  subscriber      => subscriber,
  rule            => 'tab.user_data.reason_id = 9 and
tab.user_data.object_name = 'TEST_TBS'' );
END;
/

```

**Example 2: Tablespace full exception subscription**

```

DECLARE
  subscriber      aq$_agent;
BEGIN
  subscriber := aq$_agent('RECO_AREA_SUB','ALERT_QUEUE',null);
  dbms_aqadm.add_subscriber(
    queue_name     => 'ALERT_QUEUE',
    subscriber     => subscriber,
    rule           => 'tab.user_data.reason_id = 130' );
END;
/

```

**Example 3: Recovery area alert subscription**

2. Next, create a callback procedure that contains the remedy for the exceptions. For example, the remedy could involve running the segment advisor and implementing its recommendations when a particular tablespace is running out of space. Sample code for a PL/SQL callback procedure for the subscriber SPACE\_SUB is shown below.

```
create or replace procedure myCALLBACK(
  context raw, reginfo sys.aq$_reg_info, descr sys.aq$_descriptor,
  payload raw, payloadl number)
AS
  dequeue_options    DBMS_AQ.dequeue_options_t;
  message_properties DBMS_AQ.message_properties_t;
  message_handle     RAW(16);
  message            ALERT_TYPE;
BEGIN
  dequeue_options.consumer_name := 'SPACE_SUB' ;

  -- Dequeue the message
  DBMS_AQ.DEQUEUE(queue_name => 'SYS.ALERT_QUE',
                  dequeue_options => dequeue_options,
                  message_properties => message_properties,
                  payload => message,
                  msgid => message_handle);

  commit;
  -- provide your remedy PL/SQL code here, e.g., run Segment Shrink.
  commit;
END;
/
```

3. The third and final step is to register the callback procedure. The following sample code shows how the procedure myCALLBACK is registered to subscriber SPACE\_SUB.

```
DECLARE
  reginfo1    sys.aq$_reg_info;
  reginfo1list sys.aq$_reg_info_list;
BEGIN
  reginfo1 := sys.aq$_reg_info('SYS.ALERT_QUE:SPACE_SUB',
                              DBMS_AQ.NAMESPACE_AQ, 'plssql://SYS.MYCALLBACK',
                              HEXTORAW('FF'));

  -- Create the registration info list
  reginfo1list := sys.aq$_reg_info_list(reginfo1);

  sys.dbms_aq.register(reginfo1list, 1);
END;
/
```

Every time an alert is generated to which the user has subscribed, a corresponding remedy will be applied automatically via the callback procedure, depending on the type of alert. Thus, Oracle Database provides two ways in which a self-corrective action can be taken: using Oracle Enterprise Manager's response actions, and through Oracle Advanced Queuing's callback procedure capability. This makes it possible to create truly self-reliant systems.

### **Remedies for Commonly Encountered Exceptions in Oracle Database 11g**

The following remedies can be incorporated into Oracle Enterprise Manager response actions or Oracle Advanced Queuing's callback procedures.

#### **Space-Related Exception Handling**

The most common space problems involve tablespaces that are running out of space. Oracle9i Database introduced the resumable space allocation feature, which prevents space-related problems from aborting transactions, by instructing the database to suspend any operation that encounters an out-of-space condition. This provides an opportunity for corrective action to be taken while the operation is suspended, and the operation automatically resumes its execution once the problem has been resolved.

In Oracle Database 11g, a warning alert is generated when a tablespace is 85 percent full and a critical alert is generated when it is 97 percent full (these default values for the thresholds are user-modifiable). Thus, anytime a session that someone has marked as resumable—by issuing the `ALTER SESSION ENABLE RESUMABLE` command in the session—encounters an out-of-space condition, the session goes into a suspend mode and a space alert is automatically generated. A response action can be defined for the particular tablespace in Oracle Enterprise Manager that runs a script to remedy the problem. The remedy can include one or both of the following actions:

- Use online segment shrink to recover wasted space from the tablespace in question. This could fix the problem without addition of a new datafile. The following SQL command shrinks the segment specified:

```
ALTER TABLE <table_name> SHRINK SPACE;
```

- Add a new datafile to the tablespace. By use of the Oracle-managed files (OMF) feature discussed earlier, this script can be made fairly generic without the need to hard-code the actual name, location, or size of the datafile.

#### **ORA-1555 (“Snapshot Too Old”) Error Handling**

Oracle Database 11g also generates an alert when ORA-1555 is encountered. For such an alert, the corrective action involves enabling automatic undo retention tuning and increasing the size of the undo tablespace. For more-sophisticated handling, the undo advisor can be called via the PL/SQL package `DBMS_UNDO_ADV` and its output used to determine the exact amount of

space needed. The following anonymous PL/SQL block shows a very simple example of how the undo advisor can be used to determine the correct size of the undo tablespace.

```
declare
    retention number ; undo_ts_size number ;
begin
    retention := DBMS_UNDO_ADV.REQUIRED_RETENTION ;
    undo_ts_size := DBMS_UNDO_ADV.REQUIRED_UNDO_SIZE(retention) ;
    dbms_output.put_line('Required tablespace size is ' || undo_ts_size);
end ;
/
```

In this way, a script can be created that determines the actual size of the undo tablespace needed to remedy ORA-1555, and the size of the undo tablespace can then be altered by addition of a new datafile. This corrects the exception condition and the transaction or query can then be resubmitted for successful completion, thus obviating the need for external intervention.

#### **Recovery Area Alert Handling**

This alert is generated in Oracle Database 11g when the flash recovery area encounters space pressure because there are no more files that can be deleted and the archiver or Oracle RMAN needs to write something to it. The appropriate response action can include one or more of the following remedies:

- Allocate a larger space quota to the flash recovery area.
- Back up files to tape or some other tertiary device.
- Reduce the retention policy, such as with the Oracle RMAN `configure` command:

```
RMAN CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF <x> DAYS;
```

- Merge incremental backups into the image copies, using the following command:

```
RECOVER COPY OF DATAFILE <datafile# or name>
```

This will allow the deletion of incremental backup files.

These actions remedy problems such as databases hanging because redo logs cannot be archived, or regular backups failing because of lack of space in the recovery area.

Oracle Database 11g provides the basic infrastructure for designing self-reliant systems. With minimal effort, some of the more common types of problems can easily be handled. This infrastructure is key to making the database truly self-reliant in a way that is virtually transparent to the application user.

## Diagnosing and Resolving Problems

An embedded database must provide an easy way to diagnose and resolve problems when they do occur. Eventually even a flawless embedded database installation will encounter problems that require administrative intervention. Granted, these cases should be rare in an embedded database, but the database must nevertheless provide an easy way to detect, diagnose, and resolve problems if and when they do happen.

The vast majority of organizations with embedded installations do not have skilled in-house database system technical expertise, so it is even more essential that the process of diagnosing and fixing problems be straightforward enough that even a novice user can do it easily. To this end, Oracle Database 11g has self-diagnostic capabilities that enable it to proactively detect and diagnose problems.

### Automatic Database Diagnostic Monitor

The automatic database diagnostic monitor (ADDM) is an Oracle Database feature that provides a holistic, real-time performance picture of the entire database. It runs proactively every 30 minutes and looks for any performance-related problem. It can also be run on demand anytime. After every run, it generates a report containing all of its findings. The findings highlight sources of problems as well as nonproblem areas. If a problem is detected, it will identify its root causes and make appropriate recommendations for solving it. Anytime it encounters a database performance problem that requires user intervention, the first and only thing the user needs to look at is the ADDM report. The report lists the problem and recommends a solution, along with its performance benefit.

ADDM has a PL/SQL and Oracle Enterprise Manager interface. ISVs and OEMs can use the PL/SQL interface to embed the ADDM capabilities into their application or use the Oracle Enterprise Manager interface to remotely diagnose their customers' databases. They can also use the ADDM PL/SQL interface to automate the submission of ADDM reports to themselves so they can provide proactive support to their end users.

The DBMS\_ADVISOR package is provided in Oracle Database 11g to assist in using the PL/SQL interface. Following are three examples of how the DBMS\_ADVISOR package can be used from the command line.

The following example shows how to manually run ADDM from SQL\*Plus.

```
set serveroutput on;

begin
  declare
    id number; name varchar2(100); descr varchar2(500);
  BEGIN
    name := ' ' ; descr := 'manual addm run';
```

```

    dbms_advisor.create_task('ADDM',id,name,descr,null);
    dbms_advisor.set_task_parameter(name, 'START_SNAPSHOT', 420);
    dbms_advisor.set_task_parameter(name, 'END_SNAPSHOT', 421);
    dbms_advisor.set_task_parameter(name, 'INSTANCE', 1);
    -- execute task
    dbms_advisor.execute_task(name);
end;
end;
/

```

The next example shows a SQL\*Plus script for seeing the latest ADDM report.

```

SET LONG 1000000
SET PAGESIZE 0
SET LONGCHUNKSIZE 1000
SET VERIFY OFF
COLUMN get_clob format a80

select dbms_advisor.get_task_report(dal.task_name, 'TEXT', 'ALL')
from   dba_advisor_log dal
where  dal.task_id = (select max(t.task_id)
                    from   dba_advisor_tasks t, dba_advisor_log l
                    where  t.task_id = l.task_id
                    and    t.advisor_name = 'ADDM'
                    and    l.status = 'COMPLETED');

```

The final example shows a SQL\*Plus script for seeing an ADDM report for a specific ADDM task. The script is invoked from SQL\*Plus as "@<Script\_Name> <TASK\_NAME>".

```

SET LONG 1000000
SET PAGESIZE 0
SET LONGCHUNKSIZE 1000
SET VERIFY OFF
COLUMN get_clob format a80

select dbms_advisor.get_task_report('&1', 'TEXT', 'ALL') from dual;

```

ADDM provides a very easy, straightforward way for even novice users to diagnose and resolve all performance-related problems. The power and ease of use of this feature make it essential for embedded database installations when those rare performance problems requiring user intervention do occur.

## Conclusion

Embedded databases have many benefits, not only for ISVs and OEMs developing a business application, but also for their end users or customers. Embedded databases enable ISVs and OEMs to build critical technology into their products for higher cost savings, improved management control, and greater efficiency. For end users, embedded databases offer high performance and rapid installation with lower IT and support costs.

The technical challenges involved in an embedded database are numerous. Installation and configuration must be completely invisible to users and fully integrated with their application. The embedded database must be self-managing, with all routine administrative tasks automated and common problems addressed without outside intervention. Ongoing maintenance and support must also be automated and streamlined.

Oracle Database 11g has a variety of features and enhancements that aid in the deployment, self-management, and ongoing maintenance of embedded databases. Oracle Database 11g enables a truly embeddable database that is easy to deploy; that automates all routine administrative and management tasks; that offers easy, self-managing problem diagnostics and remedies; and that automates software upgrades and patches.



The Self-Managing Database: Deploying Oracle  
Database 11g in Embedded Environments  
April 2009

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

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



| Oracle is committed to developing practices and products that help protect the environment

Copyright © 2009, Oracle and/or its affiliates. All rights reserved. This document is provided for information purposes only and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.