E
Loading and Reading

This appendix covers a variety of topics extending those covered in *Chapter 7, Techniques for Creating a Multimedia Database*. It includes in greater details the performance tests discussed on tablespace fragment sizes. It covers multimedia table creation statements and PL/SQL code for performing loads. Also detailed is how to install and configure Apache 2.0 with the Mod PL/SQL gateway. The final section reviews sizing methods that can be used to configure Oracle databases of various sizes and hardware capabilities.

Locally managed tablespace's UNIFORM extent size

The following testing results show how the block size and extent size can impact performance.

Disclaimer: The tests results shown here were done locally and do not represent any official results. They have not been independently verified (the goal is to have this done with the release of part 2 of this book). The results should be seen as a guide. A database administrator reviewing these results should confirm the same behavior at their own site before attempting to embark on large-scale projects using these results as a proof.

The scripts used in this testing procedure are found at the end of this section. The test involved creating a tablespace with varying `BLOCKSIZE` and `LOCAL UNIFORM SIZE` using the command:

```
CREATE TABLESPACE tbls_name BLOCKSIZE 8192/16384 EXTENT MANAGEMENT
LOCAL UNIFORM SIZE xx segment space management auto datafile
'directory/datafile' size 6G reuse;
```

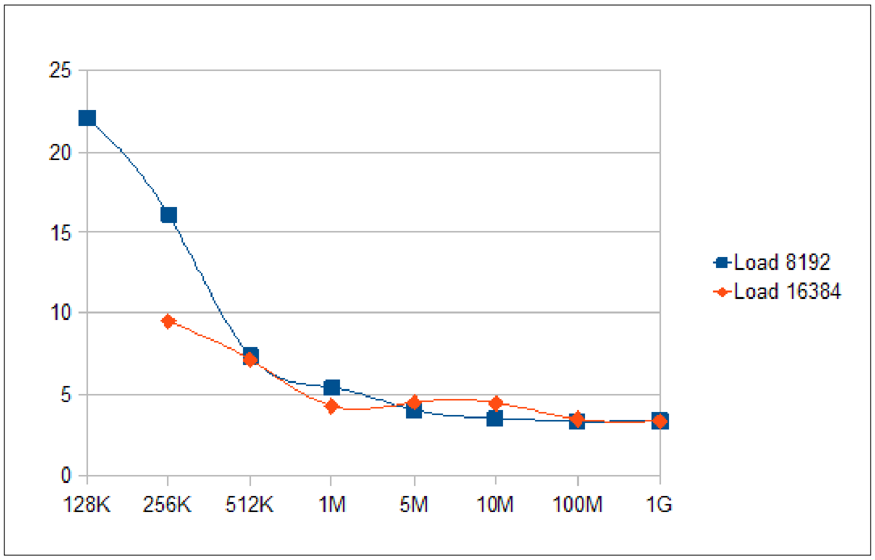
67 TIF images of various sizes between 10 MB and 90 MB are loaded in (no processing at all), totaling in size of approximately 3.0 GB. Times are taken for the load to run. A read test is then done where each BLOB is read in chunks of 32 KB and the size in bytes calculated. This ensures the database has to perform a complete full scan of the BLOB. As the BLOB is not cached, it is read in from disk. The test was run three times with the best result used as optimal. For an 8 KB block size, the minimum extent size is 128 KB, whereas for 16 KB block size, it's 256 KB. This is because `segment space management auto` was used and requires a minimum number of extents to be created.

The following table lists the testing results comparing an 8-KB block size with a 16-KB block size for loading in digital images:

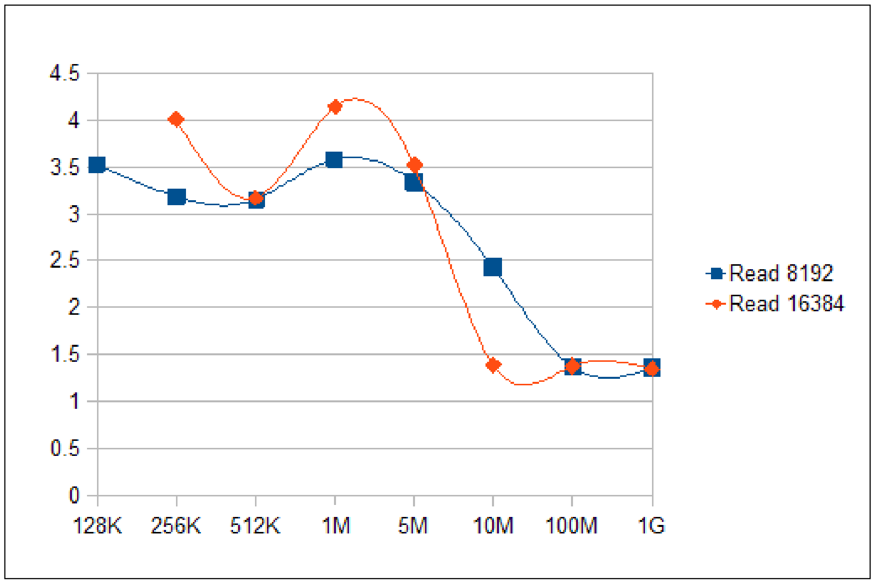
LOCAL UNIFORM SIZE	**BLOCKSIZE 8192**			**BLOCKSIZE 16384**		
	Load	**Extents**	**Read**	**Load**	**Extents**	**Read**
128 KB	22.08 min	27,241	3.53 min	n/a	n/a	n/a
256 KB	16.13 min	12,831	3.19 min	9.50 min	13,179	4.01 min
512 KB	7.37 min	6,309	3.16 min	7.17 min	6,366	3.17 min
1 MB	5.45 min	3,129	3.59 min	4.29 min	3,130	4.14 min
5 MB	4.05 min	622	3.34 min	4.52 min	618	3.52 min
10 MB	3.55 min	311	2.43 min	4.50 min	309	1.39 min
100 MB	3.35 min	32	1.37 min	3.49 min	31	1.38 min
1 GB	3.41 min	4	1.36 min	3.37 min	4	1.35 min

The results raise some interesting issues. There is obviously a sweet spot around the 10 to 50 MB mark for extent size. Once it has reached the database, block size becomes immaterial, and this is due to the number of extents dropping to under 800. The test results clearly show that an extent size under 1 MB in size will result in dramatically slower load times as well as slower read times.

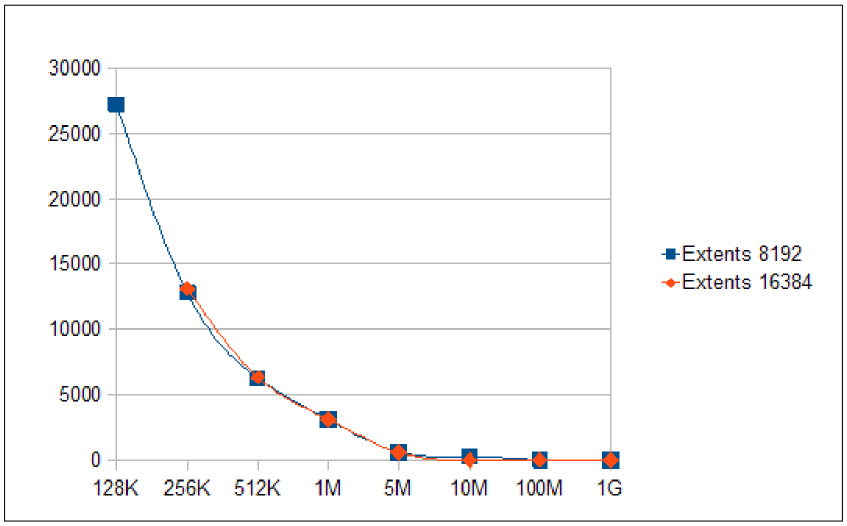
The following graph plots the load results shown in the previous table:



The following graph plots the read results shown in the previous table:



The following graph plots the extent usage results shown in the previous table:



The recommendation is, if the decision is made to use an 8 KB block size, the extent size for the tablespace should be a minimum of 10 MB. Tablespace storing large volumes of storage should use a large extent size with the aim of keeping the number of extents below 1,000. Using a 16 KB block size will not immediately improve performance.

Test Server Specifications: The test server used was deliberately designed to be the one which is mid-range using standard off the shelf components. Servers with a higher specification are likely to get faster results. The goal of the testing wasn't to determine the best performance capabilities, but rather to compare features and see which database capabilities work best with multimedia.

Operating system: Windows 2008R2

Oracle Database Version: 11.2.0.2 – 64 Bit

DB cache size: 300 MB

PGA size: 300 MB

Java Pool: 300 MB

Shared Pool: 300 MB

CPUs: 1 x Intel Core i7-2600 CU @ 3.40 Ghtz

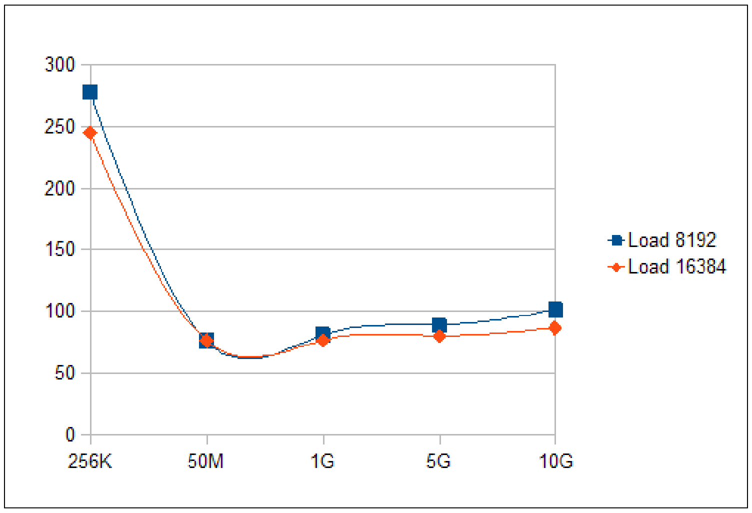
If there is some confusion about why the number of extents is approximately the same between the 8 KB and 16-KB block size, when one size is double the size of the other, keep in mind that the extent size is a fixed size which is configured when the tablespace is created. Using the 1 GB uniform extent size, even though both 8 KB and 16 KB have four extents, 8 KB requires 524,288 blocks to achieve it while 16 KB only requires near half that at 262,144 blocks.

The hypothesis test case is that the sweet spot for image loading is about 10 to 50 MB of uniform extent size based on the notion that the number of extents should be no more than about 800. If this hypothesis holds true, then increasing the volume of data by 100 times should show that the 10 to 50 MB of sweet spot now becomes a choke point, and the ideal extent size becomes 1 to 5 GB, with no real improvement if the extent size increases beyond this. An alternate view is that the processing based on extents is an approximate fixed length of time and is exacerbated by the smaller volume size. As the data volume increases, this size will remain static and for very large volumes prove to be negligible.

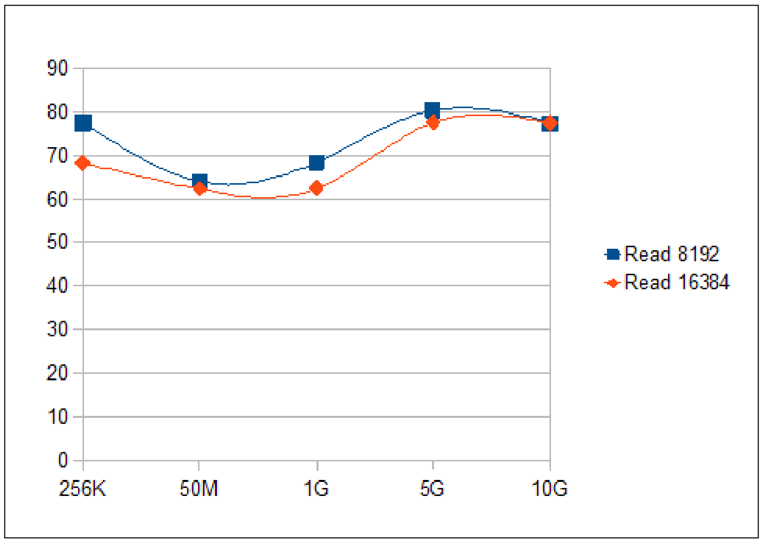
In the next test, as covered in the following table, the volume of data processed is increased by 100 fold. Some basic tuning has been done on the database to improve load times (ensure redo is on a separate I/O channel and larger redo log sizes). The following table shows the load and read times when processing nearly 300 GB of images:

LOCAL UNIFORM SIZE	**BLOCK SIZE 8192**			**BLOCKSIZE 16384**		
	Load	**Extents**	**Read**	**Load**	**Extents**	**Read**
256 KB	278.40 min	1,228,477	74.56 min	245.08 min	1265633	68.35 min
50 MB	76.53 min	5,972	64.11 min	76.48 min	5920	62.35 min
1 GB	81.36 min	292	68.56 min	77.03 min	290	62.52 min
5 GB	89.23 min	59	80.40 min	84.37 min	59	77.52 min
10 GB	101.39 min	31	77.47 min	87.30 min	30	77.53 min

The following graph plots the load results of the previous table:



The following graph plots the read results of the previous table:



In comparison to the smaller loads, extending the size by 100 has shown some interesting storage issues. As expected, the smaller extent size impacted the load time; it did not impact the read time. The database is happily navigating the bitmap header and retrieving the blocks in good time.

The extent size sweet spot is still being seen with loads. A size of from 100 MB to under 1 GB is optimal; compared to the first test with a smaller load size, where the optimal extent size was around 10 MB.

The astute reader would notice that for a larger extent size, from 5 GB onwards, the read time actually increased by a little amount. This effect was not seen in the initial test, though a read bump was seen at around the 1 MB to 5 MB mark in it. Repeated tests confirmed the bump.

In an attempt to understand why this was happening, an additional test was done to see what would happen if automatic segment space was set to manual. The disadvantage with this test (as covered in the code section at the end of the chapter) is that `SECUREFILE` option only works when segment space management is automatic, resulting in the test having to use the older `BASICFILE` option for creating LOBs.

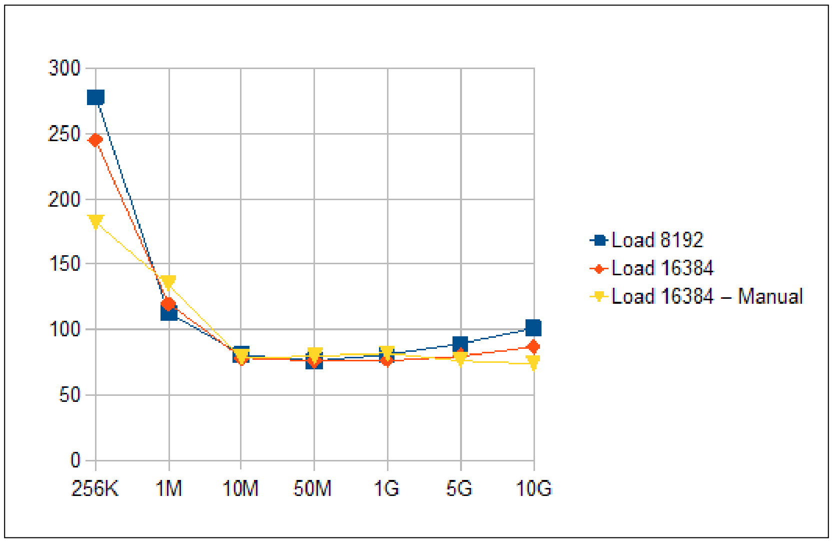
The tablespace for storing the `BASICFILE` option was created as follows:

```
CREATE TABLESPACE tbls_name BLOCKSIZE 16384 EXTENT MANAGEMENT LOCAL
UNIFORM SIZE xx segment space management manual datafile 'directory/
datafile' size 6G reuse;
```

The load results were as follows:

Write 300 GB			
`LOCAL UNIFORM SIZE`	`SECUREFILE`		`BASICFILE`
Block Size	8192	16384	16384
Space Manage	Auto	Auto	Manual
256 KB	278.4	245.8	182.57
1 MB	113.13	120.16	135.09
10 MB	81.38	78.52	79.24
50 MB	76.53	76.48	80.1
1 GB	81.36	77.03	82.05
5 GB	89.23	80.4	77.01
10 GB	101.39	87.3	74.15

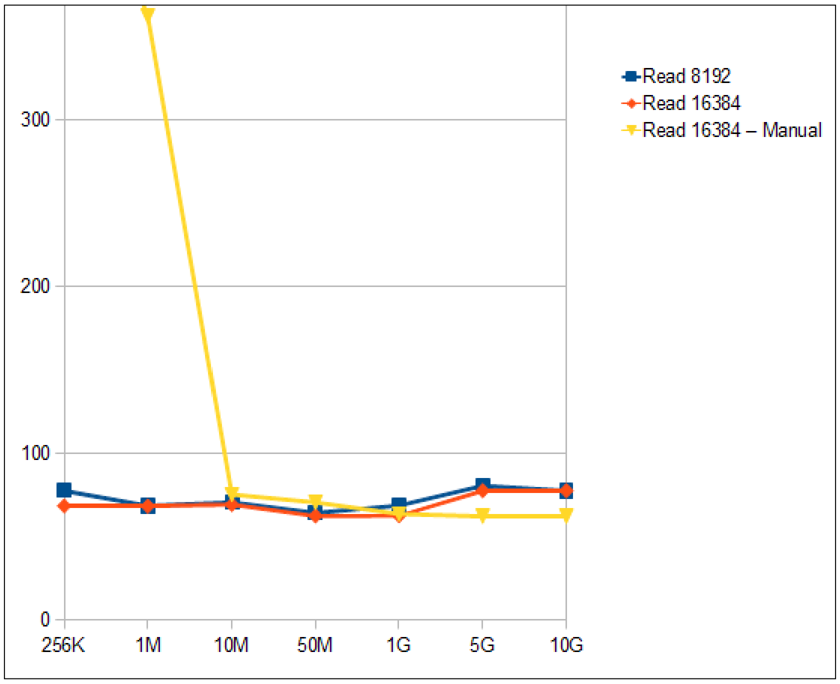
The following graph plots the load results of the previous table:



The following table details the results of performing reads of the data:

Read 300 GB			
LOCAL UNIFORM SIZE	SECUREFILE		BASICFILE
Block size	**8192**	**16384**	**16384**
Space manage	**Auto**	**Auto**	**Manual**
256 K	77.56	68.35	499.18
1 M	68.43	68.22	362.12
10 M	70.48	69.19	75.19
50 M	64.11	62.35	70.45
1 G	68.56	62.52	63.52
5 G	80.4	77.52	62.17
10 G	77.47	77.53	62.09

The following graph plots the read results of the previous table:



There was a marked difference in performance for `BASICFILES` when the extent size was small, showing how much more consistent and faster `SECUREFILES` is. For the data load size, the previous graph shows that a 100 MB extents size using a 16 KB block size is optimal for `SECUREFILES`.

As a comparison test, the statement using `EXTENT MANAGEMENT LOCAL AUTOALLOCATE` was done on a block size of 16,384, with the following results:

```
WRITE: 72.46
READ: 61.41
```

The previous results show that the `AUTOALLOCATE` clause is more efficient than using a `UNIFORM EXTENT` size.

An additional test was done to load in 300 GB, but this time choosing the optimal extent size derived from the previous tests. This involved loading in 6 x 50 GB images using `dbmslob.loadfromfile`, which is a supplied PL/SQL Package command. The results showed that the `AUTOALLOCATE` clause proved very efficient and slightly ahead of the 100 MB extent size. Though `BASICFILES` performed well with a large extent, its restrictions and scalability as well its impending de-support, rule out its usage. Refer to the following table:

	Securefile		**Basicfile**			
	Autoallocate	**100 MB Extent**	**100 MB Extent**	**1 GB Extent**	**10 GB Extent**	**50 GB Extent**
Load (Write)	71.51	72.18	81.16	67.14	67.42	72.08
Read	63.17	64.08	68.38	63.56	64.18	66.41

A final comparison test was done to determine how efficient the `DEDUPLICATE` option is. This option checks to see if the same image will be loaded in twice and, if so, only stores it once. As covered in *Chapter 7*, *Techniques for Creating a Multimedia Database*, the goal with this option is to save on storage. There is no expectation of improvement in load or read times. As it is reloading in 100 times, the same 67 set of images that only consumed 3 GB of storage, the total storage used should be very close to the storage consumed in the first test. Time will be required to load the image in, but savings in time will be gained, because no writes will be needed to store it, using tablespace creation parameters of:

```
BLOCKSIZE 16384 EXTENT MANAGEMENT LOCAL UNIFORM SIZE 100 M
```

The load time was of 87 minutes and the read time was of 63 minutes. The storage was dramatically reduced indicating that the `DEDUPLICATE` option works:

```
select blocks,extents,(bytes/1024)/1000 "Mb"
from user_segments
where segment_name = 'L_TEST_LOAD';

    BLOCKS    EXTENTS         Mb
---------- ---------- ----------
    198400         31     3174.4
```

Normally, when the results don't meet expectations, it either means that the test is wrong, the hypothesis is wrong, or there is something invalid about the database and further testing using different extent sizes is required.

Code used for test load and reading

The following section covers methods for creating multimedia tablespaces and creating objects based on Oracle multimedia types.

Tablespaces

The following code snippet shows an example for creating a locally managed tablespace, with the automatic space management enabled (tables were created using the `SECUREFILE` option):

```
CREATE TABLESPACE tbls_name BLOCKSIZE 16384 EXTENT MANAGEMENT LOCAL
UNIFORM SIZE 5G segment space management auto datafile 'directory/
datafile' size 50G;
alter tablespace tbls_name add datafile 'directory/datafile' size 50G;
```

When the block size was reduced from 16,384 to 8,192, the following error occurred:

```
ORA-01144: File size (6553600 blocks) exceeds maximum of 4194303
blocks
```

The first solution was to create the physical file size at 25 GB and not 50 GB.

The second solution involves using the `BIGFILE` option in tablespace creation:

```
CREATE BIGFILE TABLESPACE tbls_name BLOCKSIZE 8192 EXTENT MANAGEMENT
LOCAL UNIFORM SIZE 10G segment space management auto datafile
'directory/datafile' size 50G;
```

The limitation with `BIGFILE` is that it can only contain one datafile, which means it can only reside on one disk system. If the storage exceeds the size of the datafile, it can be increased in size using the following query:

```
alter database datafile 'directory/datafile' resize 100G;
```

The following code shows an example for creating a tablespace with the space management set at manual. Only tables created using the `BASICFILE` option can be made using this structure. 400 GB of storage was added in 50 GB sets of datafiles.

```
CREATE TABLESPACE tbls_name BLOCKSIZE 16384 EXTENT MANAGEMENT LOCAL
UNIFORM SIZE 1G segment space management manual datafile 'directory/
datafile' size 50G;
alter tablespace tbls_name add datafile 'directory/datafile' size 50G;
```

Trying to create a table using the `SECUREFILE` option in a tablespace with manual storage management will result in the following error:

```
ORA-43853: SECUREFILE lobs cannot be used in non-ASSM tablespace
```

Tables

The following is the table creation statement used in the load tests:

```
create table test_load
 (
   pk        number(16),
   vimg      ORDSYS.ORDIMAGE,
   stl       timestamp,
   edl       timestamp
)
tablespace tbls_name_relational_data pctfree 0 storage
( pctincrease 0 maxextents unlimited)
  LOB (vimg.source.localdata) STORE AS SECUREFILE l_test_load
     (TABLESPACE tbls_name disable storage in row
      RETENTION AUTO
      NOCOMPRESS
      KEEP_DUPLICATES
      STORAGE (MAXEXTENTS UNLIMITED PCTINCREASE 0 )
     NOCACHE LOGGING);
```

For the test, where automatic space management was disabled, the table was created using `BASICFILE` with the following storage definition:

```
create table test_load
 (
   pk        number(16),
   vimg      ORDSYS.ORDIMAGE,
   stl       timestamp,
   edl       timestamp
)
tablespace PICTION_MED_1 pctfree 0 storage( pctincrease 0 maxextents
unlimited)
  LOB (vimg.source.localdata) STORE AS BASICFILE l_test_load
     (TABLESPACE PICTION_IMG_2 disable storage in row
      STORAGE (MAXEXTENTS UNLIMITED PCTINCREASE 0 )
       CHUNK 16384
       NOCACHE LOGGING);
```

Procedures

The following procedure is used to load an image located on the filesystem into the database. The procedure takes in the physical filename as input. It's assumed that all images are located in one physical directory.

```
create or replace procedure ldimg( fname varchar2 )
as
```

```
 cursor c1( vpk integer ) is
     select * from test_load where pk = vpk for update;

  srec     test_load%ROWTYPE;
  ctx     RAW(4000) := NULL;

begin
 select s_object.nextval into srec.pk from dual;
 srec.stl := systimestamp;
 srec.vimg := ordsys.ordimage.init();
 insert into test_load values srec;
 commit;
 open c1( srec.pk );
 fetch c1 into srec;
 close c1;
 srec.vimg.setSource('FILE', 'LOADING_DIR', fname );
 srec.vimg.import(ctx);
 srec.edl := systimestamp;
 update test_load set row = srec where pk = srec.pk;
 commit;
 update test_load set edl = systimestamp where pk = srec.pk;
 commit;
end ldimg;
/
```

The following is an example of code used to load a set of images in calculating the total time the load took. In this example, only five images are shown, but all tests used a base of 67 images:

```
truncate table test_load;
declare
 st timestamp;
 ed timestamp;
begin
 st := systimestamp;
ldimg('myimage1.tif');
ldimg('myimage2.tif');
ldimg('myimage3.tif');
ldimg('myimage4.tif');
ldimg('myimage5.tif');
 ed := systimestamp;
 dbms_output.put_line( 'Start = ' || st );
 dbms_output.put_line( 'End = ' || ed );
 dbms_output.put_line( 'Diff = ' || (ed - st) );
end;
/
```

To load in 300 GB of image data, the 67 images were repeatedly reloaded in just by enclosing the `ldimg` procedures in a for loop.

Optimizer statistics were immediately calculated once the routine finished its run.

The following is a simple routine for reading in and processing all the data in a table and ensuring every BLOB is read in. It's designed to see how long it takes to read all binary data in. For checking purposes, it returns the time to run and the total size processed.

```
declare
 v_count  integer;
 v_int    integer;
 r_buffer  raw(32767);
 sz    integer;
 st timestamp;
 ed timestamp;
begin
 st := systimestamp;
 sz := 0;
 for c1rec in (select * from test_load) loop
  v_count := 0;
  v_int := 32767;
  loop
   begin
    dbms_lob.read(c1rec.vimg.source.localdata,v_int,(v_
count*32767)+1,r_buffer);
    v_count := v_count + 1;
    sz := sz + nvl(utl_raw.length(r_buffer),0);
   exception when others then exit;
   end;
  end loop;
 end loop;
 ed := systimestamp;
 dbms_output.put_line( 'Start = ' || st );
 dbms_output.put_line( 'End = ' || ed );
 dbms_output.put_line( 'Diff = ' || (ed - st) );
 dbms_output.put_line( 'Tot = ' || sz );
end;
/
```

Manually configuring access to a BFILE

The following anonymous block shows how to manually set and define a BFILE, enabling Multimedia to reference an image located on a filesystem. When used, the BLOB will be ignored:

```
declare
  cursor c1 is select vimg from test_load where pk = 1 for update;
  drec ORDSYS.ORDIMAGE;
begin
 insert into test_load(pk,vimg)
         values(1,ordsys.ordimage.init());
 commit;
 open c1; fetch c1 into drec; close c1;
 drec.source.srclocation := 'LOADING_DIR';
 drec.source.srcname := 'myimg.tif';
 drec.source.srctype := 'file';
 drec.source.local := 0;
 drec.setproperties;
 dbms_output.put_line('Width x Height = ' || drec.width || 'x' ||
drec.height);
 dbms_output.put_line('Length = ' || drec.contentlength);
 dbms_output.put_line('Mimetype = ' || drec.mimetype);
end;
/
```

The output produced is as follows:

```
Width x Height = 2809x4176
Length = 35216308
Mimetype = image/tiff
```

Note that if `drec.source.local` is set to a value of `1`, then an error is returned:

```
ERROR at line 1:
ORA-29400: data cartridge error
IMG-00701: unable to set the properties of an empty image
ORA-06512: at "ORDSYS.ORDIMG_PKG", line 1149
ORA-06512: at "ORDSYS.ORDIMAGE", line 220
ORA-06512: at line 12
```

This is because Oracle Multimedia is looking for the data in the BLOB column (`source.localdata`), and there is no data there. Using a value of `0` forces Multimedia to use the `BFILE` structure to locate the external file.

Loading in an image from a HTTP location

The following section provides an example of PL/SQL code which could be used to load in digital images from the filesystem.

```
declare
  cursor c1 is select vimg from test_load where pk = 1 for update;
  drec     ORDSYS.ORDIMAGE;
  ctx    raw(4000);
begin
 insert into test_load2(pk,vimg) values(1,ordsys.ordimage.init());
 commit;
 open c1; fetch c1 into drec; close c1;
 drec.importfrom(ctx, 'http', 'www.mysite.com',
      'icons/images/admin/loader.png');
 drec.setproperties;
 dbms_output.put_line('Width x Height = ' || drec.width ||
       'x' || drec.height);
 dbms_output.put_line('Length = ' || drec.contentlength);
 dbms_output.put_line('Mimetype = ' || drec.mimetype);
 dbms_output.put_line('srctype - local = ' ||
       drec.source.srctype || ' - ' || drec.source.local);
end;
/
```

The code when executed produced the following:

```
Width x Height = 16x16
Length = 770
Mimetype = image/png
srctype - local = http - 1
```

Consider the following error code:

```
ORA-24247: network access denied by access control list (ACL)
```

If the previous error is returned, then it means that the schema has not been configured to use TCP to reference outside data sources. See the next section for how to do this.

Rather than loading the image, it can still be referenced externally similar to `BFILE`. The following shows how this can be manually done (the image is not loaded into BLOB but is just referenced instead):

```
declare
  cursor c1 is select vimg from test_load where pk = 1 for update;
  drec ORDSYS.ORDIMAGE;
begin
 insert into test_load(pk,vimg)
         values(1,ordsys.ordimage.init());
 commit;
 open c1; fetch c1 into drec; close c1;
 drec.source.srclocation := 'www.xor.com.au';
 drec.source.srcname := 'icons/mimsy/nstmnu/loader.png';
 drec.source.srctype := 'http';
 drec.source.local := 0;
 drec.setproperties;
 dbms_output.put_line('Width x Height = ' || drec.width || 'x' ||
drec.height);
 dbms_output.put_line('Length = ' || drec.contentlength);
 dbms_output.put_line('Mimetype = ' || drec.mimetype);
end;
/
```

The output produced is as follows:

```
Width x Height = 16x16
Length = 770
Mimetype = image/png
```

Example configuration for network security

The following code shows how to configure a schema to access an external TCP/IP location on any port. The schema name is `WEBSYS`, and this should be run as `SYS`:

```
BEGIN
 -- required for Oracle11 to access network
 DBMS_NETWORK_ACL_ADMIN.CREATE_ACL(
 acl => 'networkaccess.xml',
 description => 'Network permissions for *',
 principal => 'WEBSYS',
 is_grant => TRUE,
 privilege => 'connect');
END;
/
commit;

BEGIN
 DBMS_NETWORK_ACL_ADMIN.ASSIGN_ACL(
 acl => 'networkaccess.xml',
 host => '*');
END;
/
commit;

exec DBMS_NETWORK_ACL_ADMIN.ADD_PRIVILEGE('networkaccess.
xml','WEBSYS', TRUE, 'resolve');
commit;

SELECT DECODE(  DBMS_NETWORK_ACL_ADMIN.CHECK_PRIVILEGE(
'networkaccess.xml', 'WEBSYS', 'resolve'),  1, 'GRANTED', 0, 'DENIED',
NULL) PRIVILEGE  FROM DUAL;
```

How to install Apache 2.0

The HTTP server can be found at the Oracle software download site at `http://www.oracle.com/technetwork/database/enterprise-edition/downloads/index.html` under the header of **Oracle Fusion Middleware Web Tier Utilities 11g**, **Windows Download Option:**, or **Solaris Download Option:**.

As the installation is Java based, it is identical for both the Windows and Unix versions.

For those not familiar with the installation and only want to install the HTTP server, the entire installation process can be daunting. In this case, as only the HTTP server is required, the installation, if not done properly, can result in a large amount of redundant software being installed. The screen shots that follow show a safe method for ensuring that only the HTTP server is installed.

When installing, keep in mind that the installation has to go into its own Oracle kernel. So, the install must be treated like an Oracle Database install. This is different to Apache 1.3 where the install was part of the same kernel as the database.

The following installation will attempt to follow the default installation as closely as possible. These are the subsequent configuration steps for DAD, which can be shown using the default installation options.

For Windows, the assumption is that Oracle is installed in a location, such as `c:\oracle\ora11gR2`.

For Unix, the assumption is that Oracle is installed in a location, such as `/u01/oracle/ora11gR2`.

The goal in both cases is to install Apache2 into a similar directory structure.

For Windows, it will be `c:\oracle\apache2`.

For Unix, it will be `/u01/oracle/apache2`.

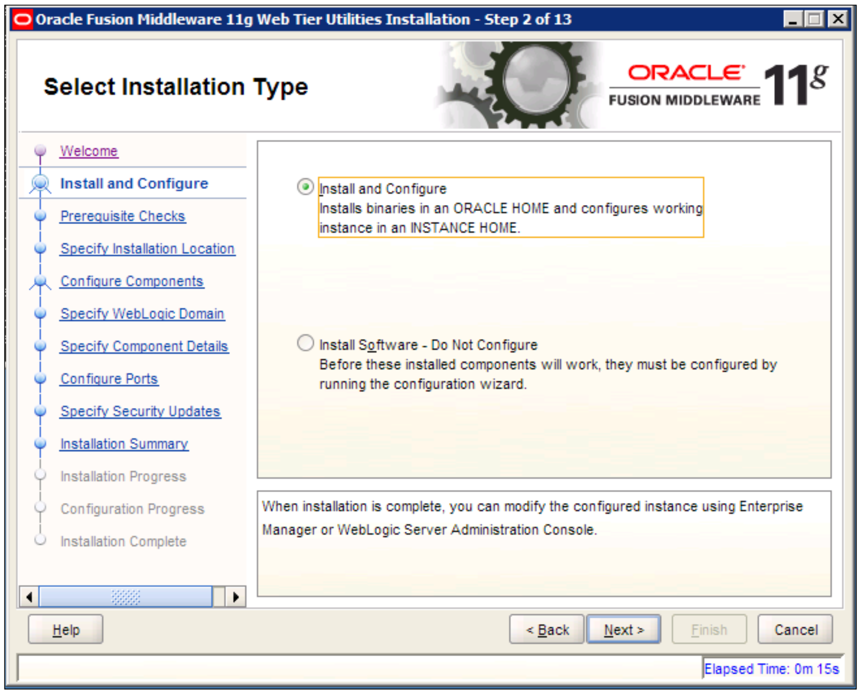
For Unix, it is suggested to avoid errors and to make it easier to debug issues; all directory names are lowercase, because Unix is case-sensitive.

Step 1 – download and unzip software

Using the steps mentioned earlier, download the Oracle software to a temporary location and unzip (unpack) the file. There will be two main sub directories created: `Disk1` and `Disk2`. Navigate to `Disk1` and run the setup file.

Step 2 – choose the installation and configure option

For novice administrators, it's best to choose **Install and Configure**. This will ensure the default installation and ports are all set up. For the HTTP server, the standard port setup is initially a good value to use.



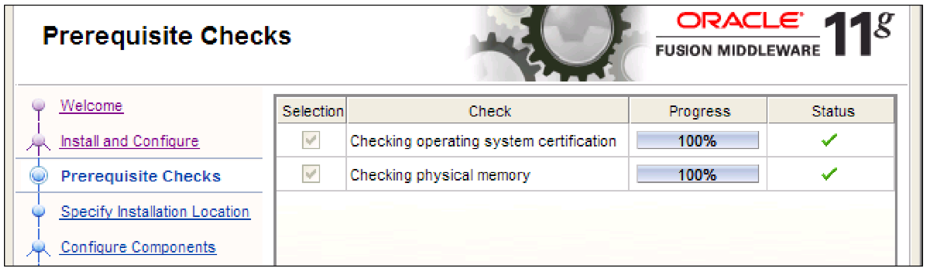
[✓ Install and Configure]
Installs binaries in an ORACLE HOME and configures working instance in an INSTANCE HOME.

Install Software - Do Not Configure
Before these installed components will work, they must be configured by running the configuration wizard.

When installation is complete, you can modify the configured instance using Enterprise Manager or WebLogic Server Administration Console.

Step 3 – prerequisite checks

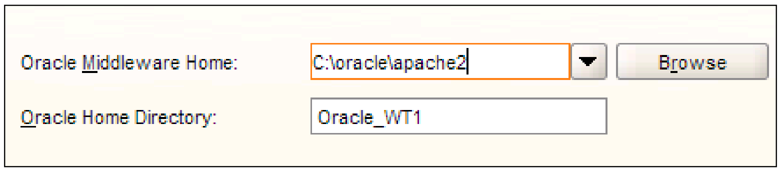
The installer will validate the server and ensure the server is correctly configured. The configuration checks differ between Windows, Linux, and Solaris.



Selection	Check	Progress	Status
✓	Checking operating system certification	100%	✓
✓	Checking physical memory	100%	✓

Step 5 – home locations

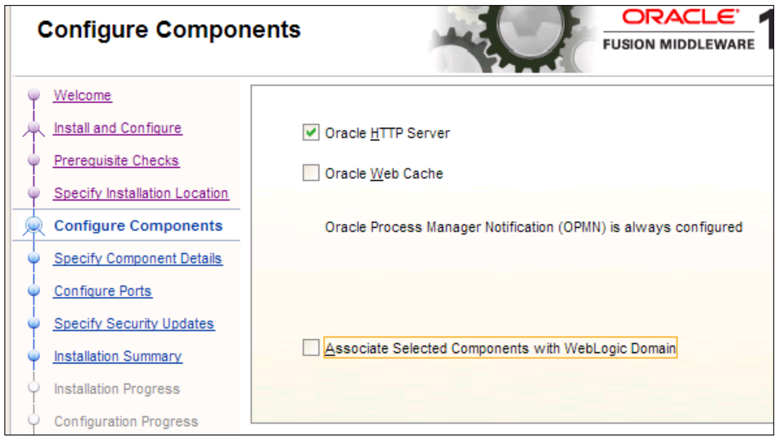
As covered earlier, choose the Oracle home location for the software.



Oracle Middleware Home:	C:\oracle\apache2
Oracle Home Directory:	Oracle_WT1

Step 6 – just install the HTTP server

Make sure the checkbox for the Oracle HTTP Server is chosen. Uncheck the other two boxes.



[✓] Oracle HTTP Server

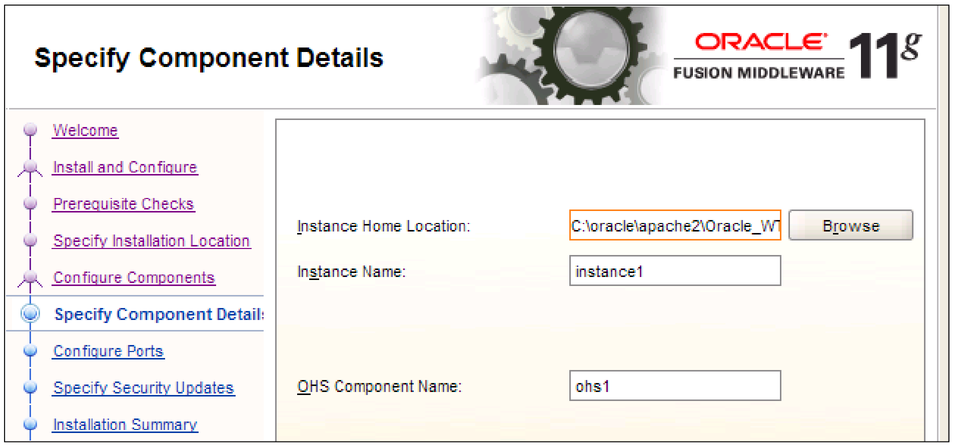
[ ] Oracle Web Cache

Oracle Process Manager Notification (OPMN) is always configured

[ ] Associate Selected Components with WebLogic Domain

Step 7 – component details

Refer to the following screenshot:

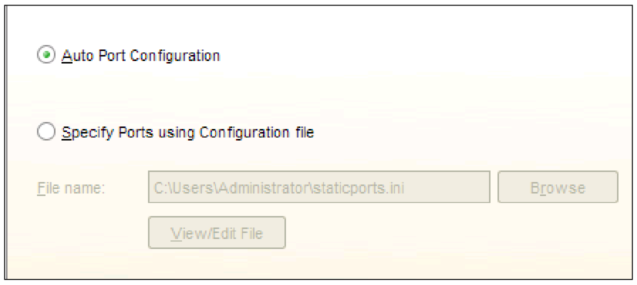


Instance Home Location: C:\oracle\apache2\Oracle_WT
Instance Name: instance1
OHS Component Name: ohs1

For windows, the directory for the main components and configuration files will be found at `C:\oracle\apache2\Oracle_WT1\instances\instance1`. The default values on the screen at this time are okay.

Step 8 – configuring autoport

The default ports on a first installation are okay. Most of the ports are for Webcache.



[✓ Auto Port Configuration] Specify Ports using Configuration file

File name: C:\Users\Administrator\staticports.ini

Step 9 – let the installer run

The next screen confirms the configuration options and performs the installation. The installation references OHS, which stands for Oracle HTTP Server. It also references OPMN, which stands for Oracle Process Manager and Notification Server. This is a command-line interface, enabling the administrator to start, stop, and manage the HTTP server.

Step 10 – post run

The HTTP server will be left running after the installation. It is listening on Port 7777. In Windows, the administrator can change the configuration file (`httpd.conf`) and change the port to the default HTTP port of 80. On Unix, only the root account can start the HTTP server when it has been modified to listen on Port 80.

Modifications made to the file require a complete restart of the HTTP server (covered later). Syntax errors or bad physical locations in the file can result in the HTTP server failing to restart.

Oracle multimedia column definitions

The following table details the column attributes within the Oracle `ORDSOURCE` type, which is located in the `ORDSYS` schema. This type is nested and referenced by the types detailed later in this section:

`ORDSYS.ORDSOURCE`		
`localdata`	`BLOB/CLOB/ BFILE`	This is the column that stores the multimedia data. Its storage definition can be referenced using `column_name.SOURCE.LOCALDATA`.
`srctype`	`varchar2 (4000)`	This is the column the BLOB is originated from. Values include file, meaning it came from the filesystem and http, meaning it was loaded from a http location.
`srclocation`	`varchar2 (4000)`	If the `srctype` is file, then the location contains the Oracle Directory name. It can also be used to contain the physical location. For a `srctype` of http, this contains the HTTP server.
`srcname`	`varchar2 (4000)`	If the `srctype` is file, then this column contains the physical filename. As an Oracle Directory can refer to at physical location, this `srcname` can include subdirectories and the filename. It can also be used to contain the physical location. For a `srctype` of http, this contains the object name.
`updatetime`	`date`	This is the column that stores date and time when the object was created or modified.
`local`	`number`	If the value is `0`, it means the object originated externally, otherwise a value of `1` indicates the data is local (see next table for examples as to how these values look based on load method).

An example of the generated values for `srctype` and `local`:

Load method	`source.srctype`	`source.local`
Physical file is loaded in from filesystem.	file	`1`
`BFILE` structure is used to point to the file residing in the filesystem; file is not to be loaded in. Example code at the end of this chapter covers a manual method of `BFILE` configuration.	file	`0`
BLOB is located in a HTTP location. Example code is included at the end of this chapter showing how to reference an external site.	http	`1`
Image is referenced using HTTP location and is not stored in the database. Example code showing how to use this is included at the end of this chapter.	http	`0`

The following table details the column attributes within the Oracle `ORDIMAGE` type, which is located in the `ORDSYS` schema. This type is used for storing digital photos. Refer to the following table:

`ORDSYS.ORDIMAGE`		
`source`	`ORDSYS.ORDSOURCE`	Column definitions are covered in *Chapter 2, Understanding Digital Objects*.
`height`	`number (38)`	
`width`	`number (38)`	
`contentlength`	`number (38)`	
`fileformat`	`varchar2 (4000)`	
`contentformat`	`varchar2 (4000)`	
`compressionformat`	`varchar2 (4000)`	
`mimetype`	`varchar2 (4000)`	

The following table details the column attributes within the Oracle ORDAUDIO type, which is located in the ORDSYS schema. This type is used for storing audio digital images.

ORDSYS.ORDAUDIO		
source	ORDSYS.ORDSOURCE	Column definitions are covered in *Chapter 2, Understanding Digital Objects*.
description	varchar2 (4000)	
format	varchar2 (31)	
mimetype	varchar2 (4000)	
comments	CLOB	
encoding	varchar2 (256)	
numberofchannels	number (38)	
samplingrate	number (38)	
samplesize	number (38)	
compressiontype	varchar2 (4000)	
audioduration	number (38)	

The following table details the column attributes within the Oracle ORDVIDEO type, which is located in the ORDSYS schema. This type is used for storing video digital images.

ORDSYS.ORDVIDEO		
source	ORDSYS.ORDSOURCE	Column definitions are covered in *Chapter 2, Understanding Digital Objects*.
description	varchar2 (4000)	
format	varchar2 (31)	
mimetype	varchar2 (4000)	
comments	CLOB	
width	number (38)	
height	number (38)	
frameresolution	number (38)	
framerate	number (38)	
videoduration	number (38)	
numberofframes	number (38)	
compressiontype	varchar2 (4000)	
numberofcolors	number (38)	
bitrate	number (38)	

The following table details the column attributes within the Oracle `ORDDOC` type, which is located in the `ORDSYS` schema. This type is used for storing digital documents.

`ORDSYS.ORDDOC`		
`source`	`ORDSYS.ORDSOURCE`	Column definitions are covered in *Chapter 2*, *Understanding Digital Objects*.
`format`	`varchar2 (80)`	
`mimetype`	`varchar2 (80)`	
`contentlength`	`number (38)`	
`Comments`	`CLOB`	

DADS.CONF parameters

The following table describes some of the parameters that can be specified in the `dads.conf` file:

Parameter	**Definition**
`PlsqlDatabaseUsername`	The Oracle username Apache has to connect to in the database. Note that a username and password does not have to be provided. If omitted, then the database resorts to digest authentication. It will prompt the user to enter in a username and password when they first access the site. This username and password is validated directly against the database and must exactly match a schema and its password in the database. Once matched, the user will continue to use that connection. The connection is still stateless.
`PlsqlDatabasePassword`	This is the unencrypted password of the schema. As discussed earlier, a `perl` program has to be run to encrypt it.

Parameter	Definition
`PlsqlDatabaseConnectString`	This can refer to the TNS entry in the `tnsnames.ora` file as can be found in this kernel. This contains information about the physical location of the database, the port number it's listening on, and the database SID to connect to. In Oracle11, shorthand syntax can be used to bypass the need for the TNS entry. In this example, Apache accesses the database locally that has `sid: orcl PlsqlDatabaseConnectString 127.0.0.1:1521:orcl`.
`PlsqlAuthenticationMode`	This refers to how Oracle is to manage connections to the database. The values it includes are basic, which is the standard method for access. No specific security program is called. Security is left up to `application.SingleSignOn`, which is used by the Oracle Single Sign On module `PerPackageOwa`, which enables a specialized function to be called to validate security before the call is made. `GlobalOwa` is similar to the `CustomOWA` one. Again it's for enabling an intercept security PL/SQL function to be called to validate the person making the call. `CustomOwa` is similar to `GlobalOwa` except the security function can be user-defined and not the one provided by Oracle.
`PlsqlDefaultPage`	This is a PL/SQL program to be called when no program is specified. For example, if the url is `http://localhost/mydad`, then Oracle, behind the scenes, appends this value to the url.
`PlsqlDocumentTablename`	This is an Oracle table used to store a file that is pushed up from the browser to the server. The table can be of any name, but it must contain a fixed set of columns with predefined names. In addition to just referencing the tablename, it can be referenced as `schema.name`, in which case the user has access to the table in another schema.

Parameter	Definition
`PlsqlSessionStateManagement`	`StatelessWithResetPackageState` (default) causes Oracle to call `dbms_session.reset_package` at the end of of each call. `StatelessWithFastRestPackageState` causes Oracle to call `dbms_session.modify_package_state` (`dbms_session.reinitialize`) at the end of each call. `StatelessWithPreservePackageState` causes Oracle to call `htp.init` at the end of the call. This maintains session variables and state but should be used with caution.
`PlsqlFetchBufferSize`	Its default value is `128`. It indicates the number of rows to be retrieved on each database call. A larger value will use more memory, use less calls, but result in a longer delay before users see the HTTP data coming back. The default is sufficient in most cases, and adjusting it does not impact the performance of retrieving multimedia objects.
`PlsqlMaxParameters`	This is the number of parameters that can be passed down in one call. The limit can be reached if there is a HTML form with a very large number of variables in it (there could be a page with 5,000 check boxes on it). Though rarely exceeded, complex HTML forms might result in it being exceeded, especially if the form includes a lot of hidden values.

Resetting the session after each call

The parameter, `PlsqlSessionStateManagement`, controls what happens to the session connected to the database when the PL/SQL call is finished. This is an important security consideration and is designed to ensure that one session cannot access another session's data. An equivalent analogy is the one regarding how an operating system maintains a virtual private state for each process.

The following table shows details regarding what each of the routines does, as mentioned earlier:

PL/SQL Reset Call	What happens in the call?
`dbms_session.reset_package`	This call frees all the memory used by the PL/SQL packages. It closes any cached cursors and ensures all package variables are reset.
`dbms_session.modify_package_state`	This call supports two calls, one of which reinitializes as covered later. This call is designed to run very quickly when freeing memory.
`dbms_session.reinitialize`	This call returns the flag value of `2`, indicating that packages are to be reinitialized without causing them to be freed from memory; cursors are closed, but not released if in cache; session memory for types and stored procedures is not freed.
`htp.init`	When Mod PL/SQL is running and when the htp package is used, the HTML returned to the user is stored in a global array. When the call is completed, the array is converted to the correct language setting and returned to the user. The `htp.init` call simply empties the array. This frees memory and ensures that the next call starts with an empty array and cannot access the contents of the previous call.

Now let's have a look at the additional Apache configuration values. In the `dads.conf` file, the following Apache values were included:

```
SetHandler pls_handler
Order deny,allow
Allow from all
```

Useful opmnctl commands

`opmnctl` is a line mode tool used to start, stop, and manage the Oracle HTTP server. It can be run from Windows DOS and any Unix shell, provided the environment variables are correctly configured. In Windows, the command is a `.bat` file, and in Unix, it's a shell script that configures environment variables before calling a perl program.

It's used behind the scenes to manage the Window Service using this command:

```
c:\oracle\apache2\opmn\bin\opmn.exe -S -I                   c:\
oracle\apache2\instances\instance1
```

The basic commands are as follows:

- This command enables all processes for the HTTP server to start:
  ```
  opmnctl startall
  ```
- This command stops the HTTP server and all processes:
  ```
  opmnctl stopall
  ```
- This command starts the HTTP server process only:
  ```
  opmnctl startproc ias-component=HTTP_Server
  ```
- This command stops the HTTP server process only. This command is useful for quickly stopping/starting the HTTP server when the `httpd.conf` file has been modified:
  ```
  opmnctl stopproc ias-component=HTTP_Server
  ```
- Alternatively, this command can achieve the same action but is much quicker:
  ```
  opmnctl restartproc ias-component=HTTP_Server
  ```

Server configuration

For database administrators, who have not worked with unstructured data or multimedia, a common question is "how should the database best be set up to manage this data?" The following sections cover a variety of different scenarios and give starting suggestions for how best to configure the database files and startup parameters. Although automatic memory management can be enabled, when working with multimedia and the fluctuating memory requirements used by it, a database administrator is in a better position if they manually tune the memory. With Oracle 10, Oracle introduced **Automatic Storage Management** (**ASM**). This option can be used but involves a different approach compared to the scenarios raised earlier. These scenarios all assume that the administrator has made the business decision not to use ASM.

Using server with only one disk and 1 GB of memory

The operating system, database kernel (software), and all database files are stored on the one disk.

In this environment, full transactional recovery in the event of media failure is not possible. If the disk storing the environment is lost, everything is lost. Depending on the storage device, some do not always fail immediately; in which case, some safeguards can be put in place in case the media starts failing but doesn't completely fail.

Backups, such as offline backups can be performed, but it's also possible to set up archiving to enable online backups. RMAN should be used to perform the backups. Even though the archives are kept on the same disk as the media, they could still prove to be useful if there is a partial disk failure. One can restore to a new disk from the backup, and hopefully, roll forward through as many archives as have survived. A good strategy might be to mirror the redo logs; just in case on disk partial failure, if a redo log is corrupted, hopefully the mirror will not be. Archiving can be enabled to achieve the same result.

If the server environment is housed within a virtualization, then snapshots can be taken. Though the snapshot will be stored on the same disk as the virtualization and not useful in the case of media failure, a snapshot can be used to roll back the entire environment in case of user error. This might eliminate the need to enable database flashback.

If there is a choice of the one storage medium that can be used, then consideration might be given to using a **Solid State Disk** (**SSD**), provided one monitors it for usage and catastrophic failure. If the database is not heavily used and is mainly for image delivery, then the disk writes will be kept low, enabling a longer life for the SSD.

Disk	Usage
Disk #1	Operating system, Oracle kernel, SYSTEM tab
Disk #1	UNDO, redo logs
Disk #1	Redo logs (mirrored), archives
Disk #1	Application tablespaces

The following are `init.ora` parameters (also referred to as the spfile) that are a good starting point for configuring memory given the constraint of 1 GB total. The figures do not add up exactly to 1 GB, and the database administrator can adjust some of these parameters until the maximum is reached.

The following table details the Oracle database memory settings for 1 GB:

Memory 1 GB	
`db_cache_size`	400 MB
`java_pool_size`	256 MB
`pga_aggregate_target`	128 MB
`shared_pool_size`	128 MB
`large_pool_size`	50 MB – if rman is used, 0 MB – if rman is not used (add 50 MB to `db_cache_size`)

Using server with two disks and 2 GB of memory

When looking at disk storage and balancing, the database administrator has three questions to answer:

- What happens if a disk fails?

  In this case, review what happens when each disk fails and what can keep running.

- How do I balance I/O load between the disks to ensure there are no bottlenecks?
- How do I ensure the storage on the disks is used optimally?

With two disks, there are a number of configuration scenarios the administrator will investigate:

- **Mirror the disks (Raid 1)**: This seems like the ideal solution, as it ensures high availability. If one disk fails, the database keeps running. Raid 1 ensures a high throughput for I/O. The first two questions raised earlier are satisfied. The downside is the third question. The total storage available is half, compared to if the disks were not mirrored. Therefore, if storage is at a premium, then this might not be the best solution. This solution also assumes that the hardware supports Raid.

- **High availability configuration**: As shown in the next table, the first disk is used to store the database and operating system files. The second disk contains sufficient information for full recovery. This solution might be required if the hardware does not support Raid 1. It is not as good as mirroring as in the event of disk failure, the whole site could fail (if the disk containing the operating system fails). Though the redo logs are mirrored and backups are kept on the second disk, ensuring no data is lost in case the first disk fails, the second disk is not fully utilized. It also doesn't ensure that there is efficient load I/O balancing, as all the work is done on the first disk. If the second disk fails, then the database will keep running until all the redo logs fill and the archiver puts the database in a hold state. The `TEMP` tablespace can be placed on the second disk. In the event of failure, it doesn't matter if it's lost, as it can be recreated.

Disk	Usage
Disk #1	Operating system, Oracle kernel, SYSTEM tab
Disk #1	UNDO, redo logs, control file
Disk #1	Application tablespaces
Disk #2	Redo logs (mirrored), archives, rman backup, control file

The administrator could go for a third solution and that involves manually balancing the database between both disks. This setup can make full use of both disks and allow the database to grow to its maximum possible size (satisfying the third question). It also improves the chances of better I/O as both disks are being utilized. By keeping backups on both disks, mirroring the redo logs across both disks, and keeping the archives on both disks ensures that in the event of disk failure, it becomes possible to fully restore the database. Any disk failure though will result in the database not working:

Disk	Usage
Disk #1	Operating system, SYSTEM tab
Disk #1	UNDO, redo logs, archives, control file
Disk #1	Application tablespaces, rman backup
Disk #2	Oracle kernel, application tablespaces, redo logs (mirrored), archives, rman backup ,TEMP tablespace, control file

If backups are done ad hoc (weekly) and stored to a portable USB drive, then it might be worthwhile for the administrator to disable archiving and configure a large number of mirrored redo logs. In this case, the administrator needs to know how many redo logs are consumed during a week and set the number and size accordingly. If 80 GB of redo is done during the week, the administrator might create 10 x 10 GB redo logs. In the event of failure, there is sufficient information in the redo logs to roll forward. It is risky, and the administrator is assuming that these redo logs are not cycled during the week, because if they are, the database cannot be fully recovered. This might be an acceptable risk for some business requirements.

So there is no ideal configuration with two disks; there are a number of options available for configuration for the database administrator, which enables them to satisfy business requirements.

The following table details the Oracle database memory settings for 3 GB:

Memory 2 GB	
`db_cache_size`	1000 MB
`java_pool_size`	256 MB
`pga_aggregate_target`	256 MB
`shared_pool_size`	256 MB
`large_pool_size`	50 MB – if rman is used, 0 MB – if rman is not used (add 50 MB to `db_cache_size`)

Using server with 3 disks and 4 GB of memory

With three disks, the database administrator has more options to work with. They can mirror the first two disks and use the third disk as a backup disk. The third disk can also be used for maintenance and contain temporary files used by the operating system or the Oracle Database (this includes trace files):

Disk	**Usage**
Disk #1/2	Operating system, Oracle kernel, SYSTEM table
Disk #1/2	UNDO, redo logs, control file
Disk #1/2	Application tablespaces
Disk #3	Redo logs (mirrored), archives, rman backup, TEMP tablespace, control file

If the hardware does not support Raid 1, then the administrator has to decide which is more important, making maximum use of the available space, or being in a position to quickly recover in the event of failure? If they need to make use of all available space, then they distribute the database files across all three disks and ensure that two disks contain mirrored redo logs, archives, and database backups.

Once configured, the database administrator should then review the whole setup and ask the following questions:

- If this disk fails, what happens?
- What is lost?
- Can the application keep running?
- What is involved in restoring the database?

The following table details the Oracle database memory settings for 4 GB:

Memory 4 GB	
`db_cache_size`	2300 MB
`java_pool_size`	256 MB
`pga_aggregate_target`	512 MB
`shared_pool_size`	512 MB
`large_pool_size`	100 MB – if rman is used, 0 MB – if rman is not used (add 100 MB to `db_cache_size`)

Using server with one disk, a SAN, and 8 GB of memory

This setup is more common today with the growth in popularity of SANs. The database administrator is given access to a very large SAN, which has its own backup and high availability options. This might involve being replicated to another site.

The problem the database administrator has is that they are unlikely going to be given a disk to do backups. The key assumption is that the SAN is completely reliable and doesn't fail.

This means that there is no requirement to perform online backups or do archiving. This assumption can prove to be dangerous if the SAN is not designed to work with Oracle. Its own block-level backups might result in the backup it performs of the database being corrupted. So, in the event of failure and restoration, the database administrator discovers they have a restored database, which can't be fully restored because of internal corruption.

The SAN could be an I/O bottleneck, but this is specific to the SAN, so the assumption to be made is that it performs well. In this case, the administrator should put on the disk any files that can be recovered using other methods. This includes the Oracle kernel, operating system, and TEMP tablespace. Anything else has to go on the SAN to ensure it can be restored.

It's possible that the disk is not even backed up, or it might be a disk residing on a virtualization and snapshots are performed on it, meaning its content and backup might be out of sync with the SAN backup. It might be worthwhile to store mirrored redo logs on the disk, just so in case of failure if the SAN can't restore fully, there might be sufficient information in the redo logs to restore. It might be then more effective to store a large amount of redo (such as 20 x 10 GB redo logs), ensuring a greater time period range that can be recovered in the event of failure.

Disk	**Usage**
Disk #1	Operating system, Oracle kernel
Disk #1	Mirrored redo Logs, control file, TEMP Tablespace
SAN	Application tablespaces
SAN	SYSTEM table, UNDO, redo logs, control file

The following table details the Oracle database memory settings for 8 GB:

Memory 8 GB	
`db_cache_size`	5 GB
`java_pool_size`	512 MB
`pga_aggregate_target`	1 GB
`shared_pool_size`	800 MB
`large_pool_size`	100 MB – if rman is used, 0 MB – if rman is not used (add 100 MB to `db_cache_size`)

Using server with a Raid structure and 16GB of memory

In this scenario, the database administrator has a number of disks and has to determine what the best Raid structure is to use to ensure high availability and performance. As covered in the discussion on Raid structures, there are a number of choices the administrator can make. If they do have choices in regards to configuration, then it is going to be a balancing act between making optimal use of storage and ensuring the best performance.

If the administrator has to configure an environment to store a large amount of multimedia and ensure high availability, and if they can configure a Raid 1+0 to store the operating system and database, this will ensure a high tolerance to failure and optimal performance. If they configure the remaining disks for Raid 5 (or 6) for storage of multimedia tablespaces, then they can achieve high reliability and efficient use by storing this larger volume of data. The assumption is that the Raid 1+0 disk system will be under 4 TB, enabling storage of all key data, but the Raid 5 system might have to grow into the tens to hundreds of terabytes range.

Usually, in an environment such as this, backups are done to tape and the requirements for recovery are configured for disaster recovery.

Disk	**Usage**
Raid 1+0	Operating system, Oracle kernel
Raid 1+0	Database files, Application tablespaces
Raid 5/6	Multimedia tablespaces (using partitioning and most are made read-only)

The following table details the Oracle database memory settings for 16 GB:

Memory 16 GB	
`db_cache_size`	10 GB
`java_pool_size`	1 GB
`pga_aggregate_target`	2 GB
`shared_pool_size`	2 GB
`large_pool_size`	10 MB – if rman is used, 0 MB – if rman is not used (add 100 MB to `db_cache_size`)

Using server with local disks, Raid, a SAN and 32 GB of memory

In this environment, the database administrator has a lot of choice about the placement of database files and operating system files; in which case, it's choosing the right disk system for the right job. As a SAN could be configured behind the scenes using Raid, the differences later assume the other capabilities a SAN can offer.

The following table highlights structures that can be targeted for each disk subsystem.

Disk	**Advantages**
Local disks	Each disk equates to one channel (when configured on the hardware correctly). It is useful for storage of multimedia images (staging), which are then loading into the database (in parallel if multiple disks are used). A local disk can also be used to store the `TEMP` tablespace (if heavily utilized), as well as mirrored redo logs.
Raid 1+0	Its advantages are high reliability and performance. It's important for storing the operating system and Oracle kernel, as in the event of disk failure, there is no impact on the environment (though performance might be impacted). It is useful for `SYSTEM`, `UNDO`, and relational based application tablespaces.
Raid 5/6	Its advantage is reliable storage for large volumes of data, especially if it's read-only.
SAN	Its advantages are high reliability, large storage capabilities, off-site mirroring, built-in backup. It is useful for multimedia tablespaces, especially ones involving a lot of read/writes. It can also be used for application tablespaces.
Solid State Disk	By keeping writes to a minimum on the SSD will ensure its long-term reliability. Very high read/write speed. It is useful for short-term redo logs storage (when doing a large volume ingest). When the operating system and Oracle kernel are stored on it, it can provide very fast restart times for the server. It is useful for storing the operating system paging/swap file. It is useful for read-only (or minimal read/write) tablespaces that are accessed frequently. This could be a tablespace containing streaming video (though sizing and cost constraints would likely limit this to the most popular video for streaming) or one containing thumbnails or most popular accessed digital images.

The following table details the Oracle database memory settings for 32 GB:

Memory 32 GB	
`db_cache_size`	17 GB
`java_pool_size`	2 GB
`pga_aggregate_target`	8 GB
`shared_pool_size`	4 GB
`large_pool_size`	100 MB – if rman is used, 0 MB – if rman is not used (add 100 MB to `db_cache_size`)

Little and big endian

When digital images are stored, numbers are routinely used to represent values in those images. This can be a pixel color or instructions for drawing a rectangle. In documents, we are used to storing numbers in their character format. The number `1089` is stored using four characters with each character corresponding to one or possibly more bytes. When storing a large number of values, this is not efficient for storage, rather the numbers are stored using bits, where each bit is either a `1` or `0`. The number 5 can be represented as `101`. When reading this value from the right-hand side to the left right-hand side, the bit corresponds to an increasing power of two. In this case, `101` is 22 + 20. The number `6` is `110` or 22 + 21.

A byte is composed of 8 bits which makes the largest number possible `255`:

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Bit**
27	26	25	24	23	22	21	20	
128	64	32	16	8	4	2	1	255

So, the `value` 255 can be stored in just one byte, where as if it was stored as characters, it would take at least 3 bytes. Larger numbers are formed by combining one or more bytes together. All numbers from `0` to `255` can be stored within 8 bits.

To make it easier to process and read, the binary values are converted into hexadecimal. A hexadecimal number has a range from 0 to 15, where the numbers 10 and above are represented as the characters from A to F. The number `10` is `A`, `11` is `B`, `12` is `C`, `13` is `D`, `14` is `E`, and `15` is `F`. The number `16` in decimal is `10` in Hex. So, the byte is split in half and each of the four half values are converted into a hexadecimal number. So, `255` becomes `FF`. A hexadecimal number is also represented using the prefix `0x`, so `255` in hexadecimal is `0xFF`. The decimal number `5` is `0x05`.

So, a large number might be represented as `5B 00 3A 00`, which is stored in 4 bytes or 32 bits. The CPU in most PCs is 32 bit, meaning the largest number they can store in an address space is 32 bits in length. The address space is used to reference disk location, memory location, file position (thus size), and registry values used for calculation. Though larger values can be stored, programming languages do not easily support them. A 64-bit CPU can store 64 bits or 8 bytes. With these CPUs now becoming standard in computers, they allow for the larger creation of files, a massive increase in the maximum memory that can be stored, and the ability to easily handle large numbers and perform calculations on them.

The problem that has occurred is that different CPU manufacturers have adopted different methods for how they read the binary values. As shown in the example earlier, the implied direction is from the right-hand side to the left-hand side, but when it comes to a value such as `5B 00`, do you start with the `5B` value first and move to `00`, or do you start with `00` and move to `5B`?

There is no right or wrong answer. However, each chip manufacturer can use the right-hand side to the left-hand side (most significant to least significant), which is also called big endian. Alternatively, they can go from the left-hand side to the right-hand side (least significant to most significant), which is also called little endian.

In the example of `5B 00`, if little endian is used, it means that the right-side value or `00` is the most significant. Most significant means the largest value component. So, as we are going up in powers of two, the right-hand value would be the largest or the most significant; in which case, the number is `91`.

In binary, the number is `0101 1011 0000 0000` (26 + 24 + 23 + 21 + 20 = 64 + 16 + 8 + 2 + 1 = 91).

So, we start with the left-hand byte and extend the number going to the right-hand side but still reading each individual bit from the right-hand side to the left-hand side.

For display we use `0101`, which is `0x05` and `1011`, which is `0x0B` (23 + 21 + 20 = 8 + 3 + 1 = 11 = Hex B).

If the number was stored in big endian, then it is read the opposite way. We take the right-hand byte and extend the number going from the right-hand side to the left-hand side:

214 + 212 + 211 + 29 + 28 = 16,384 + 4096 + 2048 + 512 + 256 = 23,296.