

SecureFiles Migration

*An Oracle White Paper
August 2008*

1. INTRODUCTION	3
2. USING SECUREFILES	3
REQUIREMENTS	3
SPECIFYING SECUREFILE.....	34
<i>LOB Storage Clause</i>	4
<i>XMLType Storage Clause</i>	4
<i>VARRAY Storage</i>	4
3. MIGRATION TECHNIQUES	4
A SIMPLE EXAMPLE	5
<i>Sample Table</i>	5
4. MIGRATION WITH ONLINE REDEFINITION	5
MIGRATING AN ENTIRE TABLE	6
MIGRATING AN ENTIRE TABLE AND ENABLING ADVANCED SECUREFILE FEATURES ..	7
ONLINE REDEFINITION OF A PARTITIONED TABLE ONE PARTITION AT A TIME	8
5. MOVING TO SECUREFILES WITHOUT MIGRATION	11
ALTER TABLE ... ADD PARTITION.....	11
CREATING PARTITIONS FROM A NON-PARTITIONED TABLE.....	12
6. CONCLUSION	13

1. INTRODUCTION

In RDBMS 8i, Oracle introduced the ability to store large unstructured and semi-structured data as Large Objects (LOBs). In 11gR1, we introduce a new LOB storage mechanism Oracle SecureFiles. It enables File system-like performance for LOBs. The old LOB storage mechanism is still available and is now called BasicFiles. In general, Oracle customers have seen performance improvements of 3-6x, with some seeing even larger gains, by migrating to SecureFiles from BasicFiles. In order to get these gains, data stored in the older (BasicFile) format needs to be migrated to the SecureFile storage format.

SecureFiles represents a completely new storage paradigm for Large Objects stored in the Oracle RDBMS. It is a complete redesign and reimplementation of LOB storage. Because of the significant changes that were required to all structures and algorithms in all levels of the RDBMS code stack to create the significant performance gains SecureFiles demonstrate, it was impossible to keep on-disk compatibility with the original LOB implementation. Thus, it is impossible to perform an “in-place” migration from BasicFiles to SecureFiles.

SecureFiles are 100% backward compatible with the various LOB APIs in the system, and migrating to SecureFiles requires no code changes to any application.

This document is intended to help Database Administrators and other technical professionals decide the best way to migrate from BasicFiles to SecureFiles.

2. USING SECUREFILES

Requirements

To use SecureFiles, the Oracle RDBMS parameter “compatible” must be set to 11.1.0.0 or higher. Also, the tablespace the SecureFile LOB segment (column/partition/sub-partition) resides in must be configured as “SEGMENT SPACE MANAGEMENT AUTO” (an ASSM tablespace).

Advanced SecureFiles features mentioned in this document may require the Advanced Compression Option or Advanced Encryption Option for use.

Specifying SecureFile

To specify that a LOB segment be stored as a SecureFile, you use the “SECUREFILE” keyword as part of the LOB storage clause. The “SECUREFILE”

keyword is tightly bound to the “STORE AS” phrase. So, in all cases where LOB storage is being specified, “SECUREFILE” will directly follow “STORE AS” and come before any other additional keywords or specifiers that might otherwise follow “STORE AS”.

Let’s look at some SQL DDL snippets to clarify this.

LOB Storage Clause

The “SECUREFILE” keyword comes after “STORE AS” and before the optional segment name:

```
LOB (c_lob) STORE AS SECUREFILE myclob
```

XMLType Storage Clause

For XMLType storage, the “SECUREFILE” keyword comes after “STORE AS” and before “CLOB” or “BINARY XML”.

CLOB storage:

```
XMLTYPE myxmltype STORE AS SECUREFILE CLOB
```

BINARY XML (BLOB) storage:

```
XMLTYPE myxmltype STORE AS SECUREFILE BINARY XML
```

VARRAY Storage

For VARRAY Storage, again, the “SECUREFILE” keyword follows “STORE AS”, and either a LOB segment name or other storage options must be specified:

```
VARRAY arr1 STORE AS SECUREFILE LOB arr1lob
```

Or alternatively:

```
VARRAY arr1 STORE AS SECUREFILE LOB (CACHE)
```

3. MIGRATION TECHNIQUES

There are several ways to migrate your applications to use SecureFiles. You can migrate entire tables to use SecureFile storage or you can migrate a partition at a time. The methods to migrate to SecureFiles are as follows:

1. Online Redefinition – We recommend Online Redefinition as the preferred method of migration as it allows the database and the table being migrated to be online during the migration process. There are two ways of migrating using Online Redefinition:
 - a. Redefine the entire table
 - b. Redefine a partition at a time

2. ALTER TABLE ADD PARTITION – If you do not have a need to migrate your existing data to SecureFiles, but want the performance going forward, you can create new partitions with SecureFile storage going forward.
3. Creating a partitioned table from a non-partitioned table – If your table currently isn't partitioned, it is possible to leave all the current data as-is and partition your table so that new data is stored as SecureFiles.

We highly recommend one of the two Online Redefinition methods of migrating to SecureFiles.

A Simple Example

Throughout this document, we will be using a simple partitioned table with one LOB column for our examples. The example is kept simple for clarity of explaining the principles behind the various ways to migrate from BasicFiles to SecureFiles. However, the prescribed methodologies should apply to all schemas.

Sample Table

```
create table table1(
  c_id number primary key,
  c_lob clob
)
lob (c_lob) store as
  (tablespace tbs_1)
partition by range(c_id) (
  partition tab1_p1 values less than (1000000),
  partition tab1_p2 values less than (2000000),
  partition tab1_p3 values less than (3000000),
  partition tab1_p4 values less than (4000000));
```

4. MIGRATION WITH ONLINE REDEFINITION

The simplest and recommended method for migrating from BasicFiles (or LONGs) to SecureFiles is to use Online Redefinition.

There are many benefits to using Online Redefinition for migration:

- 1.) The data is migrated while this system is up and available.
- 2.) When complete all LOB data is stored in the SecureFile storage format, and all the features and performance of SecureFiles are immediately available to your applications.
- 3.) Addition of features such as SecureFile Compression, SecureFile Deduplication and SecureFile Encryption can be done as the data is being migrated.

Online Redefinition requires free space equal to of the table/partition being migrated. However, migrating a single partition at a time can mitigate this.

Online redefinition uses materialized views to make the migration seamless. As the data is migrated, any changes to the source table will be picked up in the destination table. At the end of the migration, the source and destination table names are swapped so that all new updates and queries will operate against the new table. Upon completion, the destination table now holds the original data and can be dropped.

Migrating an entire table

This method works whether or not your table is partitioned. To migrate an entire table via online redefinition, you simply create a new table with the properties you want, in our case the LOB columns are SecureFiles, and migrate the data to that table. On completion, you can drop this newly created table, as it will be holding the original, pre-migration, data.

During the migration more than 2x the space is required for the original table and LOBs. This is because the data will be stored in two locations, the original table and the migration table. Upon completion of the migration, the original data can be dropped and the space can be reused. If compression and deduplication are used on the destination SecureFile columns, the amount of space required will be reduced appropriately. Please see the Administrator's Guide for more information/restrictions of online redefinition.

Let's look at a simple example of Online Redefinition of a simple table with a Single LOB column.

Our source table definition:

```
create table table1(
  c_id number primary key,
  c_lob clob
)
lob (c_lob) store as
  (tablespace tbs_1)
partition by range(c_id) (
  partition tab1_p1 values less than (1000000),
  partition tab1_p2 values less than (2000000),
  partition tab1_p3 values less than (3000000),
  partition tab1_p4 values less than (4000000));
```

Our destination table definition:

```
create table table1_int(
  c_id number primary key,
  c_lob clob
)
lob(c_lob) store as SecureFile
  (tablespace tbs_sf1)
partition by range(c_id) (
```

```

        partition tab1_sf_p1 values less than (1000000),
        partition tab1_sf_p2 values less than (2000000),
        partition tab1_sf_p3 values less than (3000000),
        partition tab1_sf_p4 values less than (4000000));
-- Note: tbs_sf1 is an ASSM tablespace.

```

Notice two significant changes to the original table definition. They are in bold above.

The tablespace is in bold to note that SecureFiles must be created in an ASSM tablespace. If you are still using MSSM tablespaces for your (BasicFile) LOB data, you will need to create or allocate ASSM tablespaces to implement the use of SecureFiles.

Performing the actual redefinition is simple. For our example, we'll assume the table above, and user "Scott".

1. Create the destination table (table1_int from above).
2. Start the redefinition:

```

dbms_redefinition.start_redef_table(
  'SCOTT', 'TABLE1', 'TABLE1_INT',
  'C_ID C_ID, C_LOB C_LOB');
-- <SCHEMA>, <Source table>, <dest_table>,
-- <Column mapping> "source.col1 dest.col1..."

```

3. Copy the table dependents:

```

dbms_redefinition.copy_table_dependents(
  'SCOTT', 'TABLE1', 'TABLE1_INT', 1, true,
  true, true, false, error_count);

```

4. Finish the redefinition (swaps the table names):

```

dbms_redefinition.finish_redef_table('SCOTT',
  'TABLE1', 'TABLE1_INT');

```

5. (Optionally) Drop the destination table:

```

DROP TABLE SCOTT.TABLE1_INT;

```

When this process completes, TABLE1's LOB column will be a SecureFile LOB column, TABLE1_INT will have TABLE1's original definition. TABLE1_INT can now be dropped to remove the space used by the original table.

Migrating an Entire Table and Enabling Advanced SecureFile Features

When performing Online Redefinition, it is possible to enable SecureFile Compression, Deduplication and/or Encryption (with the Advanced Compression and/or Advanced Encryption Options). You can do this simply by adding one or more of these options to the LOB storage clause on the destination table:

```

create table table1_int(
  c_id number primary key,
  c_lob clob
)
lob(c_lob) store as SecureFile (
  tablespace tbs_sf1

```

```

        compress low encrypt deduplicate)
partition by range(c_id) (
    partition tab1_sf_p1 values less than (1000000),
    partition tab1_sf_p2 values less than (2000000),
    partition tab1_sf_p3 values less than (3000000),
    partition tab1_sf_p4 values less than (4000000));
-- Note: tbs_sf1 is an ASSM tablespace.

```

When Online Redefinition to this destination table is complete the CLOB column will be in stored in a compressed, deduplicated and encrypted SecureFile column.

We recommend that you enable any advanced features that are desired for the resulting SecureFile data during the migration.

All of the other migration techniques discussed in this document can also have the advanced features enabled during migration.

Online Redefinition of a Partitioned Table One Partition at a Time

If you don't have the disk capacity to migrate your entire table at one time, and the table you are migrating is partitioned, you can use online redefinition to redefine each partition one at a time.

The additional space requirement here is 1x your largest partition. Thus the space required is significantly less than the space required to store your entire table and all LOB columns. Extra space is only required to hold 1 partition at a time.

In this case, we are going to operate on a partitioned table with the following definition:

```

create table scott.table1(
    c_id number primary key,
    c_lob clob
)
lob (c_lob) store as
    (tablespace tbs_1)
partition by range(c_id) (
    partition tab1_p1 values less than (1000000),
    partition tab1_p2 values less than (2000000),
    partition tab1_p3 values less than (3000000),
    partition tab1_p4 values less than (4000000));

```

The basic methodology here is to create a non-partitioned table with the SecureFile column and to migrate each partition individually. This allows us to do the migration and only requires enough excess space to store that partition's data.

The actual steps to perform the redefinition are as follows:

```

create table scott.table1_int(
    c_id number primary key,
    c_lob clob
)

```

```

lob(c_lob) store as SecureFile
  (tablespace tbs_sf1);

declare
  i number;
begin
  for i in 1 .. 4 loop
    dbms_redefinition.start_redef_table(
      'SCOTT', 'TABLE1', 'TABLE1_INT',
      NULL, dbms_redefinition.cons_use_pk,
      '', 'tab1_p' || i);
    dbms_redefinition.finish_redef_table(
      'SCOTT', 'TABLE1',
      'TABLE1_INT', 'tab1_p' || i);
    execute immediate 'drop table scott.table1_int';
    execute immediate 'create table scott.table1_int('
      || '  c_id number primary key, '
      || '  c_lob clob) '
      || '  lob(c_lob) store as '
      || '    SecureFile '
      || '    (tablespace tbs_sf1)';
  end loop;
end;
/

```

Performing the above steps will result in a table with the following definition:

```

SQL> select
dbms_metadata.get_ddl('TABLE','TABLE1','SCOTT') from
dual;

CREATE TABLE "SCOTT"."TABLE1"
(
  "C_ID" NUMBER,
  "C_LOB" CLOB,
  PRIMARY KEY ("C_ID")
USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255
STORAGE(INITIAL 16384 NEXT 16384 MINEXTENTS 1
MAXEXTENTS 505
PCTINCREASE 50 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT)
TABLESPACE "SYSTEM"
ALTER INDEX "SCOTT"."SYS_C004577" UNUSABLE ENABLE
) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
STORAGE(
BUFFER_POOL DEFAULT)
TABLESPACE "SYSTEM"
LOB ("C_LOB") STORE AS BASICFILE (
TABLESPACE "TBS_1" ENABLE STORAGE IN ROW CHUNK 8192
RETENTION
NOCACHE LOGGING
STORAGE(
BUFFER_POOL DEFAULT))
PARTITION BY RANGE ("C_ID")
(PARTITION "TAB1_P1" VALUES LESS THAN (1000000)
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255

```

```

STORAGE(INITIAL 16384 NEXT 16384 MINEXTENTS 1
MAXEXTENTS 505
PCTINCREASE 50 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT)
TABLESPACE "SYSTEM"
LOB ("C_LOB") STORE AS SECUREFILE (
TABLESPACE "TBS_SF1" ENABLE STORAGE IN ROW CHUNK
8192
NOCACHE LOGGING NOCOMPRESS KEEP_DUPLICATES
STORAGE(INITIAL 106496 NEXT 1048576 MINEXTENTS 1
MAXEXTENTS 2147483645
PCTINCREASE 0 BUFFER_POOL DEFAULT)) NOCOMPRESS ,
PARTITION "TAB1_P2" VALUES LESS THAN (2000000)
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
STORAGE(INITIAL 16384 NEXT 16384 MINEXTENTS 1
MAXEXTENTS 505
PCTINCREASE 50 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT)
TABLESPACE "SYSTEM"
LOB ("C_LOB") STORE AS SECUREFILE (
TABLESPACE "TBS_SF1" ENABLE STORAGE IN ROW CHUNK
8192
NOCACHE LOGGING NOCOMPRESS KEEP_DUPLICATES
STORAGE(INITIAL 106496 NEXT 1048576 MINEXTENTS 1
MAXEXTENTS 2147483645
PCTINCREASE 0 BUFFER_POOL DEFAULT)) NOCOMPRESS ,
PARTITION "TAB1_P3" VALUES LESS THAN (3000000)
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
STORAGE(INITIAL 16384 NEXT 16384 MINEXTENTS 1
MAXEXTENTS 505
PCTINCREASE 50 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT)
TABLESPACE "SYSTEM"
LOB ("C_LOB") STORE AS SECUREFILE (
TABLESPACE "TBS_SF1" ENABLE STORAGE IN ROW CHUNK
8192
NOCACHE LOGGING NOCOMPRESS KEEP_DUPLICATES
STORAGE(INITIAL 106496 NEXT 1048576 MINEXTENTS 1
MAXEXTENTS 2147483645
PCTINCREASE 0 BUFFER_POOL DEFAULT)) NOCOMPRESS ,
PARTITION "TAB1_P4" VALUES LESS THAN (4000000)
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
STORAGE(INITIAL 16384 NEXT 16384 MINEXTENTS 1
MAXEXTENTS 505
PCTINCREASE 50 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT)
TABLESPACE "SYSTEM"
LOB ("C_LOB") STORE AS SECUREFILE (
TABLESPACE "TBS_SF1" ENABLE STORAGE IN ROW CHUNK
8192
NOCACHE LOGGING NOCOMPRESS KEEP_DUPLICATES
STORAGE(INITIAL 106496 NEXT 1048576 MINEXTENTS 1
MAXEXTENTS 2147483645
PCTINCREASE 0 BUFFER_POOL DEFAULT)) NOCOMPRESS )

```

It is important to note here that even though the partition definitions changed, the *column* definition did not. This means that the default LOB storage for any future partitions added will be BASICFILE and **not** SECUREFILE, as is desired. You will have to explicitly specify SecureFile on any columns that you might add.

As we have demonstrated, the use of online redefinition is simple and can be performed with the database online. This method also allows you to create the new partitions with advanced options such as compression, encryption or deduplication and have the migration process also perform those transformations on the data automatically. This is why we strongly recommend this method for migrating from BasicFiles to SecureFiles.

5. MOVING TO SECUREFILES WITHOUT MIGRATION

Another possible method of migrating to SecureFiles is through the use of partitions. SecureFiles can be enabled on new partitions only, leaving the data in the older format as it is. So, new data will be stored in the high performance SecureFile format and will be able to leverage advanced features. Old data will continue to reside in the old (BasicFile) format and can be migrated at a later date, if desired.

ALTER TABLE ... ADD PARTITION

This method can be of particular use if your data is already partitioned by some monotonically increasing value (for example by date or via a sequence). In this case, you can simply define any new partitions you create to have SecureFile storage for the LOB columns.

For example, consider our simple range partitioned table:

```
create table scott.table1(
  c_id number primary key,
  c_lob clob
)
lob (c_lob) store as
  (tablespace tbs_1)
partition by range(c_id) (
  partition tab1_p1 values less than (1000000),
  partition tab1_p2 values less than (2000000),
  partition tab1_p3 values less than (3000000),
  partition tab1_p4 values less than (4000000));
```

We can add a new partition for values less than 5000000 with a simple ALTER TABLE ... ADD PARTITION:

```
ALTER TABLE table1
  ADD PARTITION tab1_p5 VALUES LESS THAN (5000000)
  LOB(col3) STORE AS SecureFile (TABLESPACE
  tbs_sf1);
```

After this is completed, all LOB data inserted with a *c_id* value between 4000000 and 4999999 will be stored as a SecureFile. This will allow you have your data going

forward be stored in SecureFiles. This single largest detractor from this is the older data still has the performance characteristics of BasicFiles since the original partitions are not migrated to SecureFiles. So, the performance characteristics of individual LOB accesses will be different across partitions. Also, the table definition didn't change, so you also must specify the LOB column is a SecureFile every time you add a partition. The advantages here are you make no significant changes to your system. Assuming the data that is in the older partitions is references less and less, your application will continue to perform better as more of the data current working set is stored in SecureFiles.

Creating Partitions from a Non-Partitioned Table

If you do not have data that is already partitioned by date or some other strictly increasing or decreasing value, then you can artificially create a partitioning scheme by creating an artificial "key" and adding that key column to the existing table before the partition exchange. Here's our simple example again, but as a non-partitioned table:

```
drop table table1;
drop table table1_sf;

create table table1(
  c_id number,
  c_lob clob
)
lob (c_lob) store as
  (tablespace tbs_1);

create sequence table1_sf_key;

create table table1_sf(
  c_id number,
  c_lob clob,
  partkey$ number default 1
)
lob (c_lob) store as SecureFile
  (tablespace tbs_sf1)
partition by range (partkey$) (
  partition bf values less than (1)
    lob(c_lob) store as BasicFile,
  partition sf values less than (MAXVALUE)
);

alter table table1 add (partkey$ number default 0);

ALTER TABLE table1_sf EXCHANGE PARTITION bf WITH
TABLE table1
  WITHOUT VALIDATION
  UPDATE GLOBAL INDEXES;

DROP TABLE table1;
RENAME table1_sf TO table1;
```

When this is complete, partition “bf” contains all the data with the LOBs stored as BasicFiles, and all data going forward will be stored as SecureFiles.

If your table isn’t partitioned, this could be an opportunity to add a more complicated partitioning scheme than the one provided here, perhaps even using the new Interval Partitioning added in Oracle RDBMS 11gR1. The key here is that whatever scheme is chosen, all the original LOB data must be fall by definition of the key partitioning into a single partition.

6. CONCLUSION

SecureFiles is a significant improvement in performance and features over the previous RDBMS LOB storage mechanism. To take advantage of the performance and new features, the LOB data must be converted to the new SecureFiles format.

For migration of existing data from BasicFiles to SecureFiles, we recommend online redefinition. This can be performed at the table or partition level as is necessary for your particular situation. It can be performed while the system is online and available to your applications. However, the space taken during the redefinition is roughly equivalent to 2x the storage required to store the current tables, LOBs and dependent objects.

If you are limited in the available disk space such that there isn’t enough space to use temporarily while the online redefinition is being performed, or if you have LOB data that is used less over time, there are other methods of migrating to SecureFiles that don’t require additional space.

ORACLE

SecureFile Migration

May 2008

Author: Scott Lynn

Contributing Authors: Vikram Kapoor, Ravi Rajamani

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

Copyright © 2008, Oracle. 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.