Storage Tiering Techniques with Oracle Database 12*c* and Oracle FS1-2 Flash Storage System

ORACLE WHITE PAPER | APRIL 2015

Table of Contents

Introduction

Storage tiering, or the act of moving data between different types of storage to match the I/O characteristics of the data to that of the storage, is a technique used by administrators for many years (decades even). The simplest and most common motivation for this is data aging. That is, as sections of data age, its access patterns change, most likely being accessed less: the data *cools* and eventually becomes *cold*. Cold data has very different I/O patterns, and hence storage needs, from warm and hot data. Even hot data may have different storage needs — data for an OLTP database workload will have a predominantly random I/O pattern and be well suited to performance solid-state drive (SSD), while the sequential workloads of decision support or analytics workloads will be better suited to capacity SSD or even hard disk drive (HDD).

Over the years, the tools and techniques available to administrators have varied, but one thing always has been the same: there is a *range* of storage media with different characteristics (price, performance, etc.). This is as much the case now as it has ever been. The specifics of the storage types change, but the fact that there is a range does not. Figure 1 shows the range of media available in Oracle FS1-2.

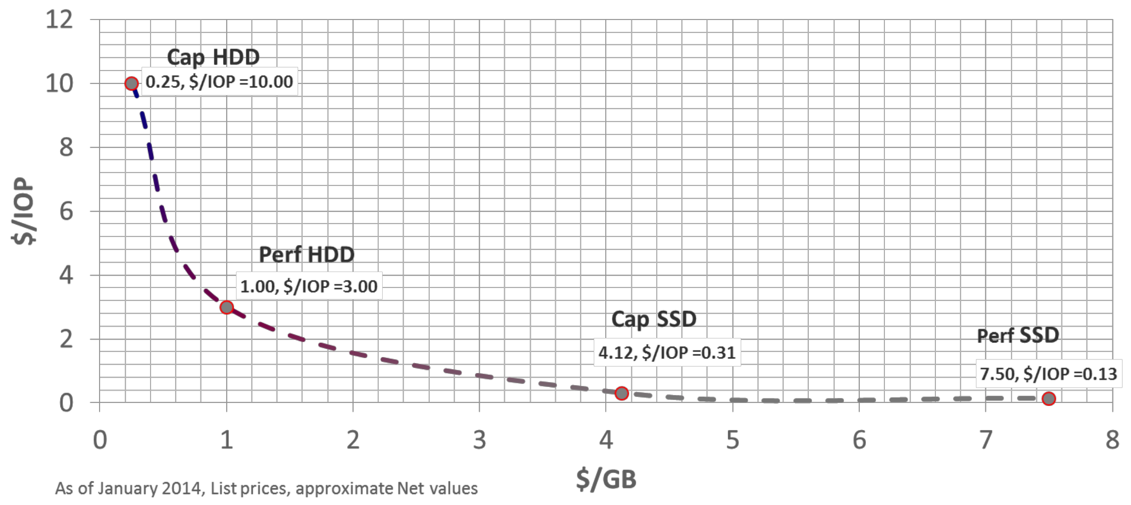


Figure 1: Price performance characteristics of different storage types

The introduction of Oracle Database 12*c* and the Oracle FS1-2 storage system adds new techniques to the administrator's arsenal. The goal of this paper is to compare and contrast those techniques. All of the techniques are useful and powerful; this paper enables the administrator to identify when each is appropriate.

This paper begins by introducing and explaining some database terminology and how this maps to storage. The main body of the paper explains the three techniques for storage tiering. The final section presents recommendations for when to use each technique.

Tables, Tablespaces, and Storage

It is necessary to understand the layers in the storage stack before delving into the various techniques for moving data within a database between storage tiers. Figure 2 shows these layers. They start with physical storage within the Oracle FS1-2 flash storage system (capacity hard drives, performance hard drives, capacity SSDs and/or performance SSDs) and move up to the volumes (virtual disks or LUNs) presented by Oracle FS1-2. Oracle Automatic Storage Management, a feature of Oracle DataBase, performs the function of a logical volume manager; it consumes the volumes presented by Oracle FS1-2 and presents them to the database as disk groups.

At the database level, the storage abstraction is a *tablespace*. A specific database has one or more tablespaces and each has one or more datafiles[1] that map to the underlying operating system. Database *tables* exist within tablespaces, and each tablespace is mapped to a specific Oracle Automatic Storage Management disk group.

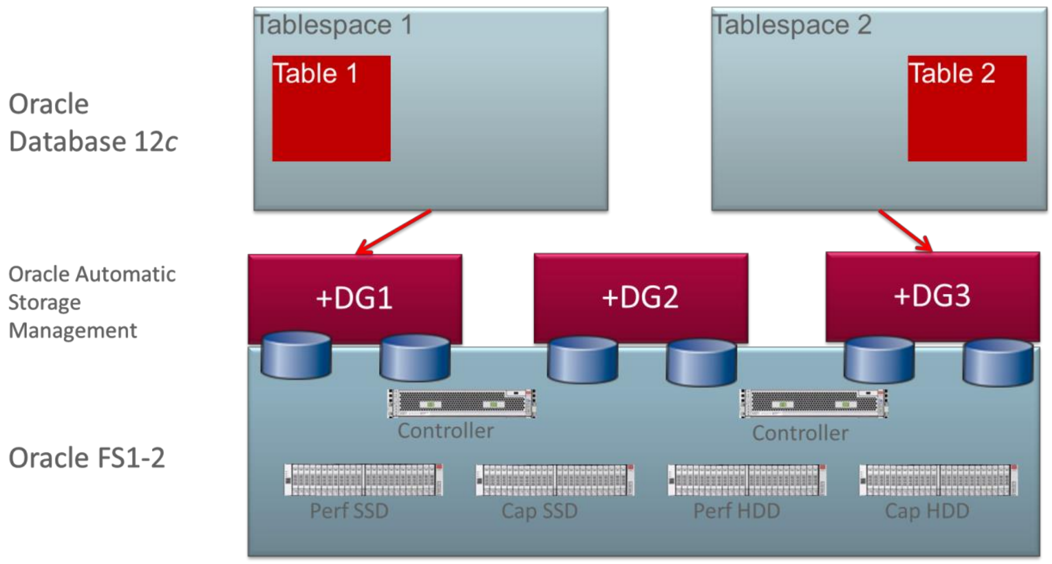


Figure 2: The relationship among tables, tablespaces, Oracle Automatic Storage Management disk groups, and storage volumes

Next is the concept of partitioning. While in Figure 2 it may be that *all* the data within a table becomes cold and, hence, the DBA wants to move the whole table to a different storage tier, much more likely is that just *parts* of the data become cold. This is where partitioning plays an important role.

[1] Oracle-managed files eliminate the need for the database administrator (DBA) to directly manage the operating system files. All operations are specified in terms of database objects rather than file names. Hence, datafiles are not discussed further in this document.

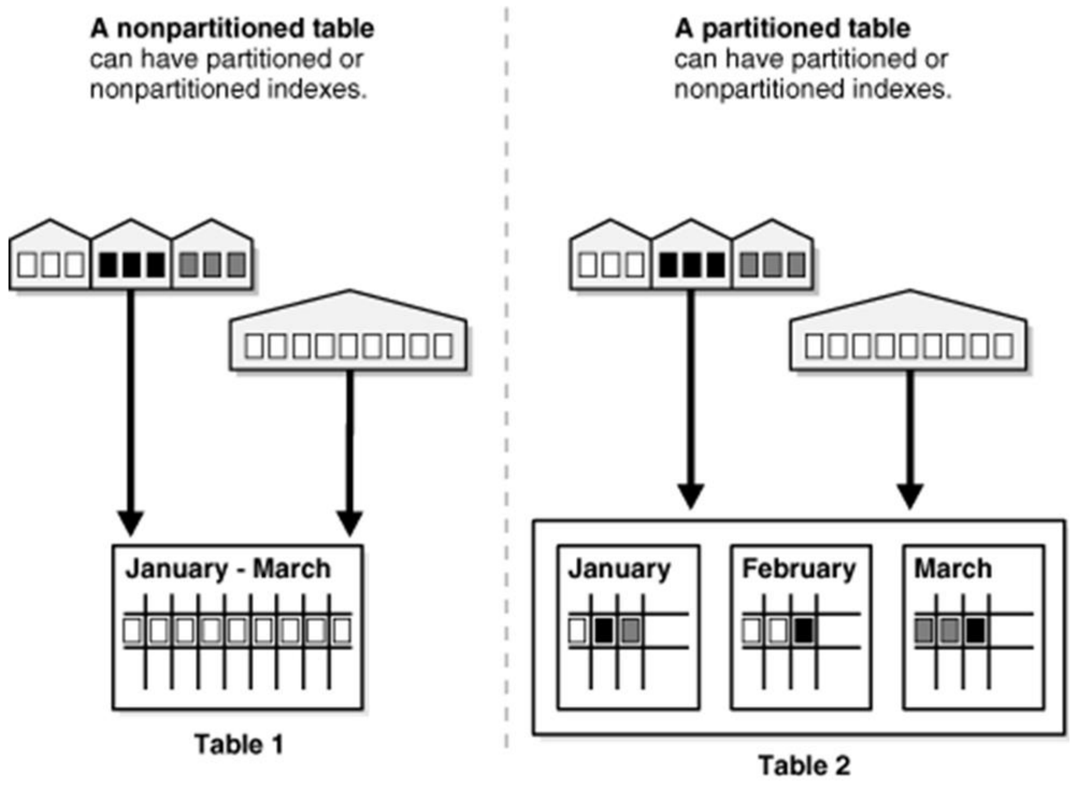


Figure 3: A view of partitioned tables

Partitioning allows a table, index, or index-organized table to be subdivided into smaller pieces, and each piece of such a database object is called a partition. Each partition has its own name, and may optionally have its own storage characteristics.

From the perspective of a database administrator, a partitioned object has multiple pieces that can be managed either collectively or individually. This gives the administrator considerable flexibility in managing partitioned objects. However, from the perspective of the application, a partitioned table is identical to a nonpartitioned table; no modifications are necessary when accessing a partitioned table using SQL queries and DML statements.

Each row in a partitioned table is unambiguously assigned to a single partition. The partitioning key is comprised of one or more columns that determine the partition in which each row will be stored. Oracle Database 12*c* *automatically* directs insert, update, and delete operations to the appropriate partition through the use of the partitioning key. The administrator has considerable flexibility about the choice of partitioning key(s), but, as Figure 3 demonstrates, by far the most common choice is based on date and/or time. For example, in an order entry OLTP database, the administrator wants to keep the current quarter's orders on high-performance storage, while migrating previous quarters' orders (now cold) to cheaper, lower-performance storage.

Figure 4 is a variant of Figure 1 in which a partitioned table is added. Table 3 has four partitions, two of which are in Tablespace 1 in disk group +DG1 and two of which have been moved to Tablespace 2 in disk group +DG3.

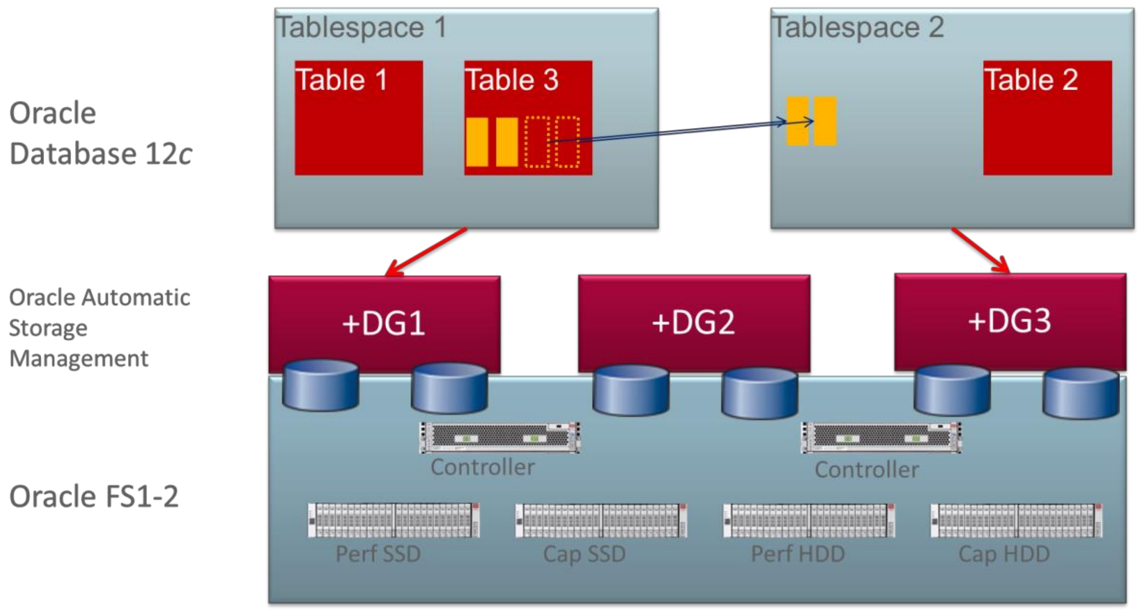


Figure 4: The relationship among tables, partitions, tablespaces, and storage

Techniques for Moving Data in Oracle Database 12*c*

In the following sections, three complementary techniques are presented:

1. DBA initiated/managed (manual)
2. Information lifecycle management (ILM) Heat Map and Automatic Data Optimization policies
3. Oracle FS1-2 Heat Map and auto-tiering

DBA Initiated/Managed

For many releases of Oracle Database, the administrator has had the ability to manually initiate the move of a table between tablespaces. Figure 5 gives a simple example.

Figure 6 gives an example of moving a table partition, rather than a whole table. Moving a table changes the rowids of the rows in the table. This causes the indexes on the table to be marked `unusable` and DML accessing the table using these indexes will receive an error; hence the need to use the `UPDATE INDEXES` clause, as shown in the figure.

```
SQL> ALTER TABLE table1
          MOVE
          TABLESPACE new_tablespace;
```

Figure 5: Example of moving a whole table to a new tablespace

```
SQL> ALTER TABLE table1
         MOVE PARTITION p1
         TABLESPACE new_tablespace
         UPDATE INDEXES;
```

Figure 6: Example of moving a table partition to a new tablespace

Moving whole tables or table partitions is simple and straightforward and can be done directly by the DBA (as shown in the simple examples of Figure 5 and Figure 6) or as a scheduled job. In the case of storage tiering, this is clearly a very useful technique for the cases where administrators know exactly *when* their data will become cold. There are many practical examples of this; for example, an administrator might know that, come the first day of each new quarter, the previous quarter's order information will no longer be accessed heavily (or at all) and hence can be moved to cheaper storage.

ILM Heat Map and Automatic Data Optimization Policies

The previous section introduced manually moved tables and partitions; useful when you know exactly when partitions are ready to move. But, what if you don't know *exactly* when your partitions are ready to move?

The example from the previous section — moving the previous quarter's partitions once the new quarter begins — is useful. But maybe orders take a few days to close. So, the orders placed in the final days of the previous quarter don't actually close until a few days into the new quarter. Maybe that's two days into the new quarter, or maybe it's even three. Without actually monitoring activity on the partition and verifying that all of its orders have gone cold, a DBA can't tell what is the right time to move that partition. This is where a new facility in Oracle Database 12*c*, the policies of a Heat Map and Automatic Data Optimization, come into play.

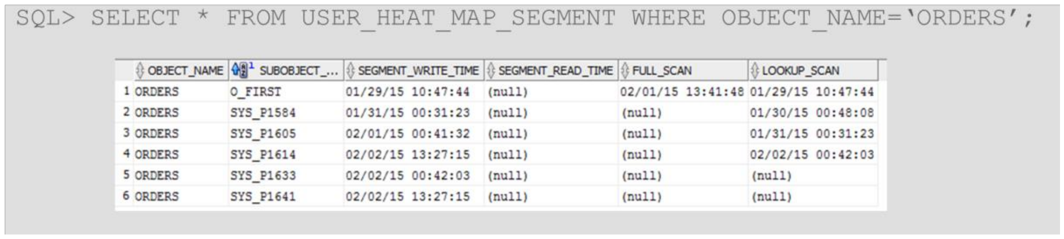
ILM Heat Map automatically tracks usage information at the row and segment levels. Data modification times are tracked at the row level and aggregated to the block level. Modification times, full table scan times, and index lookup times are also tracked at the segment level. Heat Map enables a summary view of how data is being accessed, and how access patterns are changing over time.

Heat Map must be explicitly enabled in a database — see Figure 7.

```
SQL> ALTER SYSTEM SET HEAT_MAP = ON;
```

Figure 7: Enabling Heat Map

Programmatic access to ILM Heat Map data is available through a set of PL/SQL table functions, as well as through data dictionary views — Figure 8 gives an example.



```
SQL> SELECT * FROM USER_HEAT_MAP_SEGMENT WHERE OBJECT_NAME='ORDERS';
```

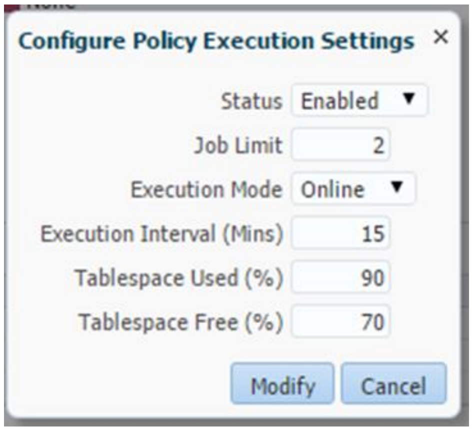
	OBJECT_NAME	SUBOBJECT_...	SEGMENT_WRITE_TIME	SEGMENT_READ_TIME	FULL_SCAN	LOOKUP_SCAN
1	ORDERS	O_FIRST	01/29/15 10:47:44	(null)	02/01/15 13:41:48	01/29/15 10:47:44
2	ORDERS	SYS_P1584	01/31/15 00:31:23	(null)	(null)	01/30/15 00:48:08
3	ORDERS	SYS_P1605	02/01/15 00:41:32	(null)	(null)	01/31/15 00:31:23
4	ORDERS	SYS_P1614	02/02/15 13:27:15	(null)	(null)	02/02/15 00:42:03
5	ORDERS	SYS_P1633	02/02/15 00:42:03	(null)	(null)	(null)
6	ORDERS	SYS_P1641	02/02/15 13:27:15	(null)	(null)	(null)

Figure 8: Using database views to access ILM Heat Map data

Automatic Data Optimization allows organizations to create policies for data compression and data movement and to implement tiering of storage; compressed or not. Oracle Database periodically evaluates Automatic Data Optimization policies, and uses the information collected by ILM Heat Map to determine which operations to execute. All Automatic Data Optimization operations are executed automatically and in the background, with no user intervention required. (Note: Because this paper is focused on storage tiering, it contains no further discussion of the compression capabilities of Heat Map and Automatic Data Optimization.) Administrators are encouraged to separately investigate and evaluate that very useful facility.

Automatic Data Optimization storage policies can be specified at the segment level for tables and table partitions. Policies are evaluated and executed automatically in the background during the maintenance window. Automatic Data Optimization policies also can be evaluated and executed anytime by a DBA, manually, or via a script. Automatic Data Optimization policies specify *what* conditions (of data access) will initiate an Automatic Data Optimization operation — such as no access, or no modification, or creation time — and *when* the policy will take effect; for example, after *n* days or months or years.

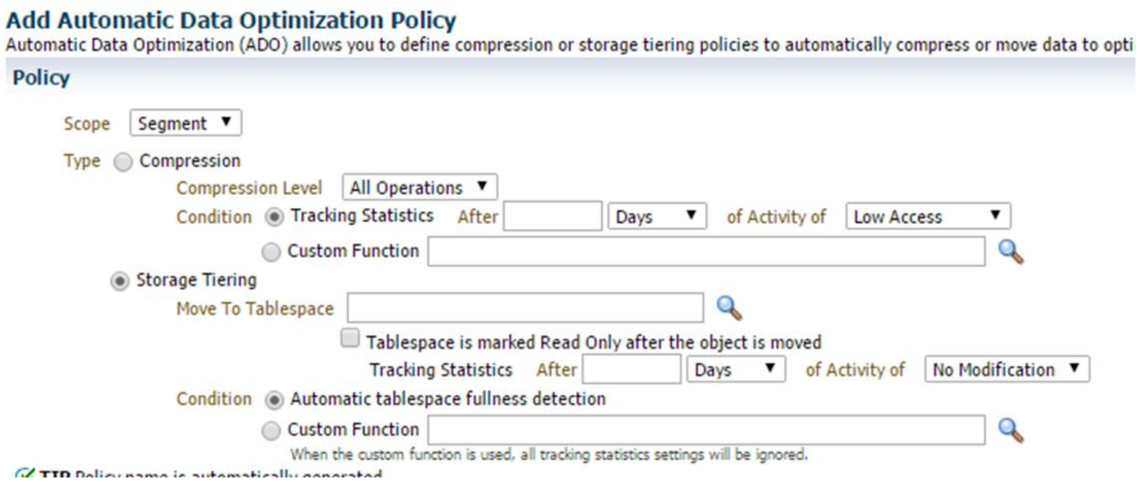
The simplest form of Automatic Data Optimization policies is triggered by tablespace usage. As Figure 9 shows, an upper limit ("Tablespace Used (%)") can be set, above which policies are triggered to move data from the tablespace in an attempt to free space; policy executions (and data movements) continue until the lower watermark ("Tablespace Free (%)") is reached. Figure 10 shows how such a policy is added by the administrator using Oracle Enterprise Manager. In this case, the DBA has elected to use a segment-level heat map (row level is not available for storage tiering policies), combined with a storage tiering policy with "Automatic tablespace fullness detection" as the execution criteria.



Configure Policy Execution Settings

Status	Enabled
Job Limit	2
Execution Mode	Online
Execution Interval (Mins)	15
Tablespace Used (%)	90
Tablespace Free (%)	70

Figure 9: Policy execution settings



Add Automatic Data Optimization Policy

Automatic Data Optimization (ADO) allows you to define compression or storage tiering policies to automatically compress or move data to opti

Policy

Scope: Segment

Type: ○ Compression

Compression Level: All Operations

Condition: ● Tracking Statistics After [ ] Days of Activity of Low Access

○ Custom Function [ ]

● Storage Tiering

Move To Tablespace [ ]

☐ Tablespace is marked Read Only after the object is moved

Tracking Statistics After [ ] Days of Activity of No Modification

Condition: ● Automatic tablespace fullness detection

○ Custom Function [ ]

When the custom function is used, all tracking statistics settings will be ignored.

Figure 10: Using Oracle Enterprise Manager to add an Automatic Data Optimization policy

Policies, such as those shown in Figure 10, are easy to configure (particularly with Oracle Enterprise Manager) and provide useful functionality. However, the real power of ILM Heat Map and Automatic Data Optimization can be exploited with custom data movement functions.

Figure 11 gives an example of such a function. The functions are coded in PL/SQL and take a single argument (an object ID). The functions must return a single value: `True` will cause data movement to occur, while `False` will inhibit it. In the example shown, the function examines `SEGMENT_WRITE_TIME` (recording the last time that the object was written to) from the `USER_HEAT_MAP_SEGMENT` (a database view of the ILM Heat Map presenting only that particular user's objects) looking for objects that have not been written to in the last three days.

Figure 12 shows the SQL command to add an Automatic Data Optimization storage tiering policy (in this case to the `orders` table) using the `move3d` function (defined in Figure 11). The function is evaluated once per day (during the designated maintenance window), and all tables or partitions for which it returns `True` are migrated to the `tier_lc` tablespace.

```
SQL> CREATE OR REPLACE FUNCTION MOVE3D
( OBJ_ID IN NUMBER )
 RETURN BOOLEAN AS
  w_time date;
  now date;
BEGIN
  select SEGMENT_WRITE_TIME into w_time
    from user_objects u join user_heat_map_segment h
    on u.object_name=h.object_name and u.subobject_name=h.subobject_name
    where u.object_name=obj_id;
  now := systimestamp;
  if (trunc(now - w_time) > 2) then RETURN TRUE; else RETURN FALSE; end if;
END MOVE3D;
```

Figure 11: Custom data movement function

```
SQL> ALTER TABLE orders ILM ADD POLICY TIER TO tier_lc ON move3d;
```

Figure 12: Adding an Automatic Data Optimization policy with a custom function

Oracle FS1-2 Heat Map and Auto-Tiering

ILM and Automatic Data Optimization are operating at the database level; that is, the ILM Heat Map is measuring and recording accesses to database objects (tablespaces, tables, partitions, etc.) and the Automatic Data Optimization policies are on database tables and partitions. This is powerful, but it means that the granularity of data movement is whole tables or partitions, typically meaning GBs or evens tens of GBs. But, what if the tables are partitioned appropriately and still have partitions with a mixture of hot and cold data? That is, the granularity of heat clusters is much smaller than a whole table or partition. This is where Oracle FS1-2's Heat Map and auto-tiering can play a valuable role.

Although both Oracle Database 12*c* and Oracle FS1-2 track data usage and create heat maps, *what* is being tracked is very different and *how* the data is used is also different. The two heat maps are compared in Table 1, but the key difference is that while ILM tracks access attributes at the table or partition level, Oracle FS1-2 tracks accesses at a subLUN granularity of 640 KB. This is, typically, much smaller than a partition and so allows the migration of much smaller chunks of data.

TABLE 1: COMPARISON OF ILM AND ORACLE FS1-2 HEAT MAPS

	ILM Heat Map	Oracle FS1-2 heat maps
Granularity	Table or partition (GBs)	640 KB
Operates on database objects	Yes (tables and partitions)	No (Block(s) of data on disk or SSD storage)
Attributes collected	Last read/write, last lookup, last scan	Access density over time, read/write ratios, random versus sequential operations, etc.
How is collected data used?	DBA-developed policies can access heat map data via database views	Administrator defines whether LUN is single tier or auto-tier. For auto-tier LUNs, FS1 system uses its internal heat map data to migrate data between storage tiers.

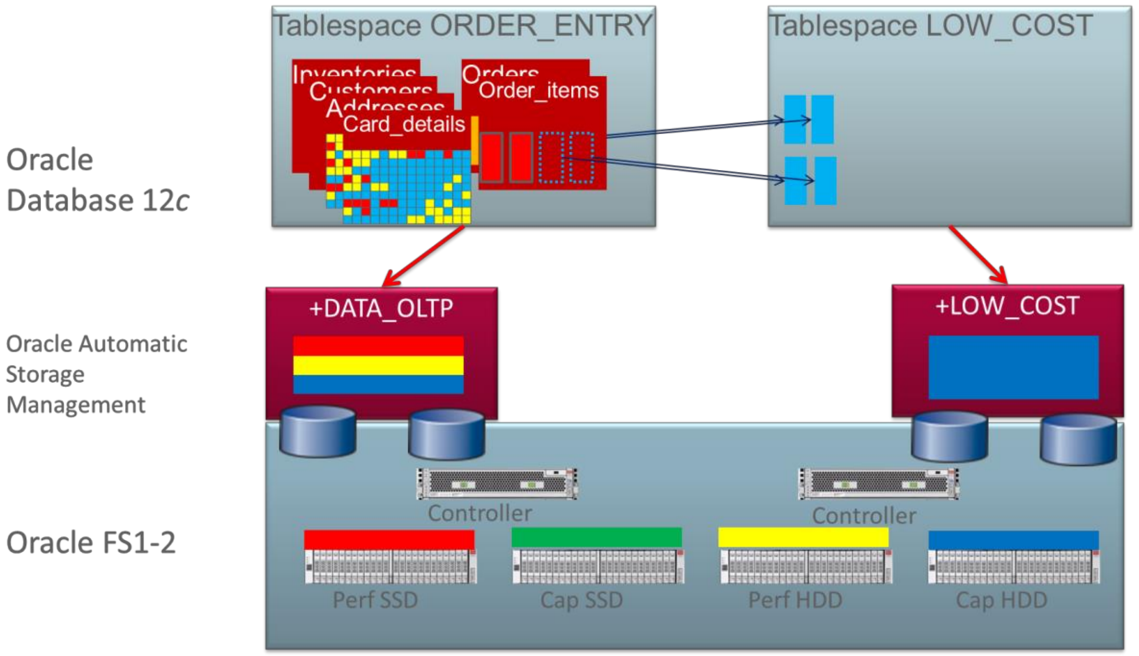
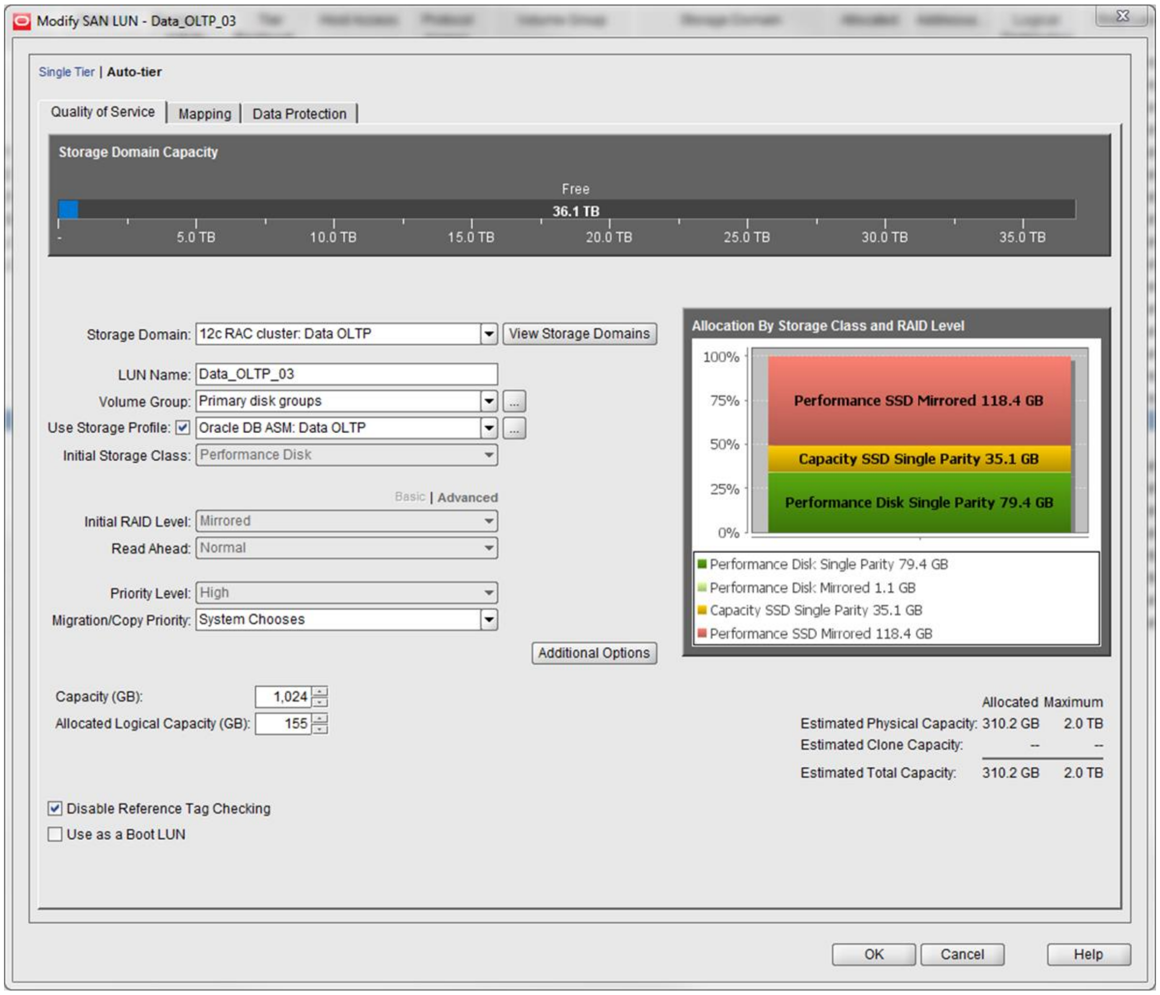


Figure 13: Using Oracle FS1-2 auto-tiering in Oracle Database 12*c*

Figure 13 takes the example previously shown in Figure 4 and extends this to include Oracle FS1-2 auto-tiering. Color coding is used to identify the different types of storage: performance SSD, capacity SSD, performance HDD and capacity HDD. There are two Oracle Automatic Storage Management disk groups:

1. `+LOW_COST` is comprised of single-tier LUNs of capacity hard drives.
2. `+DATA_OLTP` is comprised of auto-tiering LUNs comprised of performance SSD, performance HDD, and capacity HDD. Capacity SSD's could have ben configured, but for this example were not.

The same color coding is used in the database objects to indicate their heat. `Orders and Order_items` are partitioned tables where hot (recently accessed) partitions are in the `ORDER_ENTRY` tablespace (mapped to `+DATA_OLTP`), while the cold partitions have been migrated (by Automatic Data Optimization policies) to the `LOW_COST` tablespace (mapped to `+LOW_COST`). Within the other tables (for example, `Card_details`), the Oracle FS1-2 heat map shows regions of hot, warm, and cold data that have been migrated (by Oracle FS1-2 auto-tiering) to different storage tiers within `+DATA_OLTP`. These tiers are not visible to the host or the Data Base. The Data Base simply sees a LUN, so no changes are required at the DB or application level.


Modify SAN LUN - Data_OLTP_03

Single Tier | Auto-tier

Quality of Service | Mapping | Data Protection

Storage Domain Capacity

Free 36.1 TB

- 5.0 TB 10.0 TB 15.0 TB 20.0 TB 25.0 TB 30.0 TB 35.0 TB

Storage Domain:	12c RAC cluster: Data OLTP
LUN Name:	Data_OLTP_03
Volume Group:	Primary disk groups
Use Storage Profile: ✓	Oracle DB ASM: Data OLTP
Initial Storage Class:	Performance Disk
Initial RAID Level:	Mirrored
Read Ahead:	Normal
Priority Level:	High
Migration/Copy Priority:	System Chooses
Capacity (GB):	1,024
Allocated Logical Capacity (GB):	155

Basic | Advanced

Allocation By Storage Class and RAID Level

Performance SSD Mirrored 118.4 GB

Capacity SSD Single Parity 35.1 GB

Performance Disk Single Parity 79.4 GB

- 🟩 Performance Disk: Single Parity 79.4 GB
- 🟩 Performance Disk: Mirrored 1.1 GB
- 🟨 Capacity SSD Single Parity 35.1 GB
- 🟥 Performance SSD Mirrored 118.4 GB

	Allocated	Maximum
Estimated Physical Capacity:	310.2 GB	2.0 TB
Estimated Clone Capacity:	--	--
Estimated Total Capacity:	310.2 GB	2.0 TB

✓ Disable Reference Tag Checking

☐ Use as a Boot LUN

Figure 14: Screen shot from Oracle FS1-2 showing live auto-tiering results

The Oracle FS1-2 auto-tiering facility can be used to move cold data “down” to the cheapest storage, but it actually does much more: it implements a sophisticated optimization algorithm that uses the access data that it has gathered to match all of the 640 KB chunks comprising a LUN to the most appropriate storage tier. For example, it matches chunks undergoing random write behavior (an OLTP database) to performance SSD — the storage that has the best price/performance for that I/O characteristic. It also dynamically chooses an appropriate RAID level (mirrored, single parity, or double parity).

All of this is done without the DBA or storage administrator having to make explicit data movement requests. The system will automatically and transparently (without impact on database, storage or application performance) migrate chunks between storage tiers. The administrator can control the facility by explicitly turning it on or off, or by tuning the amount and types of storage in the various tiers.

Figure 14 shows the real auto-tiering results from Oracle FS1-2 that was provisioned to supply storage for an OLTP workload running on Oracle Database 12*c*. This particular auto-tier LUN is comprised of performance SSD, capacity

SSD, and performance HDD. The figure shows how the data is distributed amongst the tiers after the workload had been running for many hours. The result of all this is overall cost is lowered without sacrificing performance.

Summary and Recommendations

Manual mechanisms to move data between storage tiers in a database have been around for a long time. But recently two new techniques, information lifecycle management/Automatic Data Optimization heat maps and policies and Oracle FS1-2 heat maps and auto-tiering, have added considerable flexibility and power to the administrator's portfolio of tools. In real-life databases, all three approaches are useful and used — it isn't a matter of having to choose between them, but rather to understand the situations in which each technique is appropriate.

Here are the key recommendations:

» For the tables and partitions that go cold on a *known schedule*, the DBA can manually move the data (or create a scheduled job).

» If *all the data in a partition* goes cold together, but it's not known exactly when that will be, the ILM heat map is recommended to measure the data access patterns and create custom Automatic Data Optimization policies to undertake the data movement.

» To handle the common case where the granularity of hot/cold data is smaller than a whole partition, use Oracle FS1-2 heat map to measure the data access patterns and auto-tiering to move the data between storage tiers. This can be done independently of or in conjunction with Automatic Data Optimization policies.

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

CONNECT WITH US

blogs.oracle.com/oracle

facebook.com/oracle

twitter.com/oracle

oracle.com

Hardware and Software, Engineered to Work Together

Storage Tiering Techniques with Oracle Database 12*c* and Oracle FS1-2 Flash Storage System
April 2015