Concurrently Move DB2 for i Tables and Indexes to Solid State Disks

A simpler and more efficient way to get the most out of your Solid State Disk investment.

Mark J. Anderson June 13, 2011

Significant performance benefits can be achieved when DB2 for i indexes or tables reside on Solid State Disks (SSD). The Change Physical File Member (CHGPFM) and Change Logical File (CHGLFM) commands can now be used to concurrently move a member (or partition) of an index or table on or off SSD storage.

Overview of the DB2 for i Media Preference

Solid State Disk (SSD) support was added to IBM i in mid-2009 on releases V5R4 and IBM i 6.1. Use of SSDs drives are substantially faster than traditional disks and thus can provide performance benefits to most applications. However, the cost of SSDs was and is still higher than traditional disks, so it is imperative to provide customers with the ability to get the most out of their investment in SSDs. The way to maximize investment in SSDs is to place DB2 for i data on SSDs that will maximize the performance of your applications.

So, what data should be placed on an SSD to maximize performance?

There are three key types of data that typically are poor candidates for SSD.

1. Data that is rarely accessed. For example, if you have historical data in a table that is rarely accessed, you don't want it located on your more expensive SSD storage.
2. Data that is primarily write intensive. Even though SSD writes are faster than traditional disk writes, disk write cache typically provides good disk write performance and levels the playing field between the two types. Thus, a journal receiver would be a poor choice to put on SSD because they are primarily write-intensive.
3. Data that is primarily read sequentially. When data in IBM i is read sequentially, sophisticated pre-bring logic and buffering is used to bring data into memory in advance to its use.

The best candidate for SSD are DB2 for i objects that contain data that is frequently randomly accessed. For example, random key look-up operations against the key values stored in an SQL index or keyed logical file.

© Copyright IBM Corporation 2011

There are multiple methods to place DB2 for i objects on SSDs, but one of the most efficient is to use the DB2 media preference attribute. Media preference can be specified for an index or keyed file to request that the underlying index object be allocated on SSD storage. Media preference can also be specified on a table or physical file to request that data itself should be allocated on SSD storage. However, since the data itself is typically much larger than an index, allocating most frequently and randomly accessed indexes on SSD will likely provide the best use of the higher priced SSD storage.

Prior methods to create or move indexes or data to SSD

The initial support for SSDs in V5R4 allowed users to specify a media preference at creation or altered later with a change command as shown in the following examples:

```
CRTPF FILE(MJATST/pf1) UNIT(255)

CRTLF FILE(MJATST/lf1) UNIT(255)

CHGPF FILE(MJATST/pf1) UNIT(255)

CHGLF FILE(MJATST/lf1) UNIT(255)
```

In the IBM i 6.1 release, UNIT(*SSD) was added to the CL commands as a more usable alternative to UNIT(255). Support was also added in the IBM i 6.1 release to allow users to specify the media preference in SQL statements as shown in the following examples:

```
CREATE TABLE sales
 (unid INT primary key, salesdate DATE, itemnbr INT, quantity INT, price DECIMAL(9,2) )
UNIT SSD

CREATE INDEX salesinx on sales (unid) UNIT SSD

ALTER TABLE sales ALTER UNIT SSD
```

If the SQL table was a partitioned table, each partition (member) could have a separate media preference. This allowed users to partition the table such that frequently accessed data in one table partition could be on SSD, but less frequently accessed data could be placed in another table partition on traditional disks. For example, current data could be in a partition on SSD and historical data could be in partitions on traditional disks:

```
CREATE TABLE sales
 (unid INT primary key, salesdate DATE, itemnbr INT, quantity INT, price DECIMAL(9,2) )
 PARTITION BY RANGE(salesdate NULLS LAST)
 (PARTITION sales2006
      STARTING ('2006-01-01') ENDING('2006-12-31') INCLUSIVE UNIT ANY,
 PARTITION sales2007
      STARTING ('2007-01-01') ENDING('2007-12-31') INCLUSIVE UNIT ANY,
 PARTITION sales2008
      STARTING ('2008-01-01') ENDING('2008-12-31') INCLUSIVE UNIT SSD )

ALTER TABLE sales ALTER PARTITION sales2007
      STARTING ('2007-01-01') ENDING('2007-12-31') INCLUSIVE UNIT ANY
```

While the support for media preference in V5R4 and especially IBM i 6.1 was extensive, there were gaps:

- Media preference could be controlled at the partition (member level) for SQL partitioned tables, but the same was not possible for DDS-created physical files with multiple members.
- It was not possible to specify different media preferences on different partitions (members) of a partitioned SQL index, not to mention a DDS-created keyed logical file with multiple members.
- Lastly, all the existing statements and CL commands required an exclusive lock on the entire file.

These drawbacks have now been addressed with additions to the CHGPFM and CHGLFM commands that were also made available with the following PTFs: IBM i 6.1 (SF99601 Version 19) and IBM i 7.1 (SF99701 Version 6). There are no plans to support these enhancements on V5R4.

Using CHGPFM to move data to SSDs

The CHGPFM command will move a member's (or partition's) data either on or off of SSD. For example:

```
CHGPFM FILE(MJATST/SALES) MBR(SALES2007) UNIT(*SSD)

CHGPFM FILE(MJATST/SALES) MBR(SALES2007) UNIT(*ANY)
```

Note that if the physical file is keyed (perhaps because it has a Primary Key constraint), its index will also be moved on or off of SSD with these commands.

Using CHGPFM and CHGLFM to move indexes to SSDs

Moving indexes on or off of SSD is a bit more complicated. Indexes are created in several different situations:

- An index is created for a keyed physical or logical file. This includes both DDS-created files and SQL indexes as well as physical files with a Primary Key constraint. A CHGPFM or CHGLFM command will move the index associated with a keyed or logical file either on or off of SSD. For example:

```
CHGPFM FILE(MJATST/pf1) MBR(m1) UNIT(*SSD)

CHGLFM FILE(MJATST/lf1) MBR(m1) UNIT(*SSD)
```

- An index is created for a Unique or Foreign Key constraint. CHGPFM will not move Unique or Foreign Key indexes on or off of SSDs.
- Multiple indexes may be created for a DDS-created join logical file. CHGLFM will move all the secondary indexes associated with a join logical file on or off of SSDs.

So what do you do in the following situations?

1. You have a keyed physical file, but only want the index on SSD and not the data itself because the amount of data in the physical file is too large.

2. You have an index that is associated with a Unique or Foreign Key constraint that is heavily used in queries that you want to put on SSD.
3. You have a join logical file, but only want the index for the key on SSD, not the secondary indexes.

The way to handle each of these cases is to create simple keyed logical files or SQL indexes that share the index (or access path) you want on an SSD.

Using CHGLFM and index sharing to move indexes to SSDs

Index sharing happens when the database manager detects that you are creating an index that is the same or in some cases a subset of an index that already exists. Rather than create an entirely new index which has to be maintained, the same index is shared.

When multiple files share an index, if the media preference of any of those sharing files specifies SSD, then the index will have a media preference of SSD. If none of the files that share the index specifies SSD, then the index will have a media preference of ANY.

Assume that you have a DDS-created keyed physical file and only want the index on SSD and not the data. You can create a keyed logical file or an SQL index that shares the index of the physical file. Here's an example of an SQL Create Index statement that accomplishes the index sharing.

```
CREATE INDEX inx1 ON pf1 (key1) PAGESIZE 8
```

Then you can issue CHGLFM against the new logical file or SQL index to move the index on or off of SSD and the data will not be moved:

```
CHGLFM FILE(MJATST/inx1) UNIT(*SSD)
```

Note that if you specify the SSD attribute when you create the sharing index, the index will be moved to SSD at that time.

To get a logical file or index to share, the index attributes must be compatible. For example, an index with a LIFO attribute cannot be shared with an index that has a FIFO attribute. The following attributes must be compatible:

- The key fields and order of the key fields (sometimes a subset of the key fields is allowed)
- Ascending or descending
- Duplicate key order (UNIQUE, UNIQUE WHERE NOT NULL, LIFO, FIFO, and FCFO)
- Sort sequence or Alternate collating sequence
- Logical page size (for example, in the CREATE INDEX statement above, PAGESIZE was required)
- Maximum size (*MAX1TB, *MAX4GB, or EVI)

The easiest way to determine what the attributes of an existing index are and to determine whether the index you created actually shared an access path is to use the Show Indexes function in System i Navigator. The Show indexes task displays all the attributes of the existing index you wish to share (see Figures 1 and 2 below).

Figure 1. Launching Show Indexes for a table

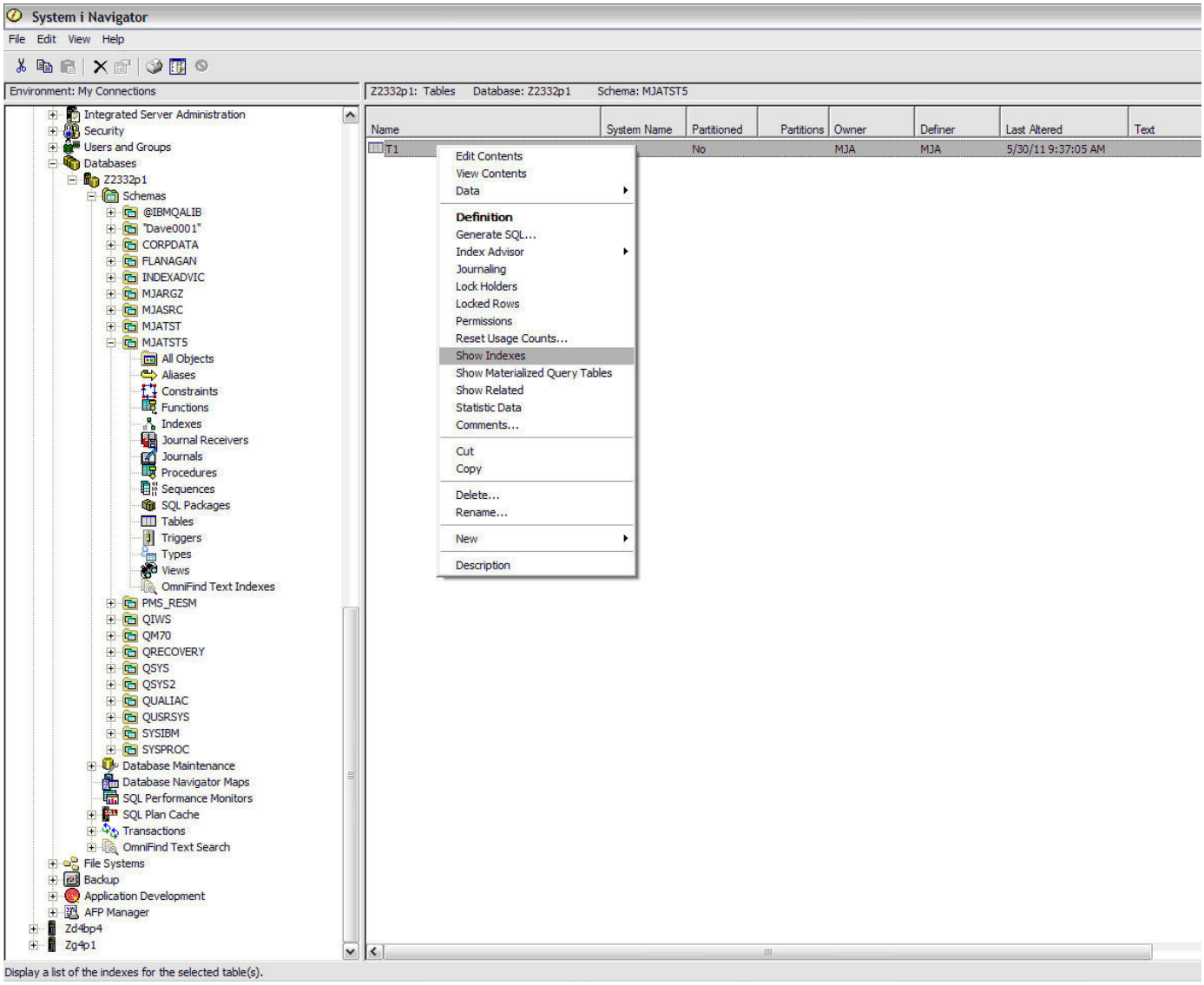
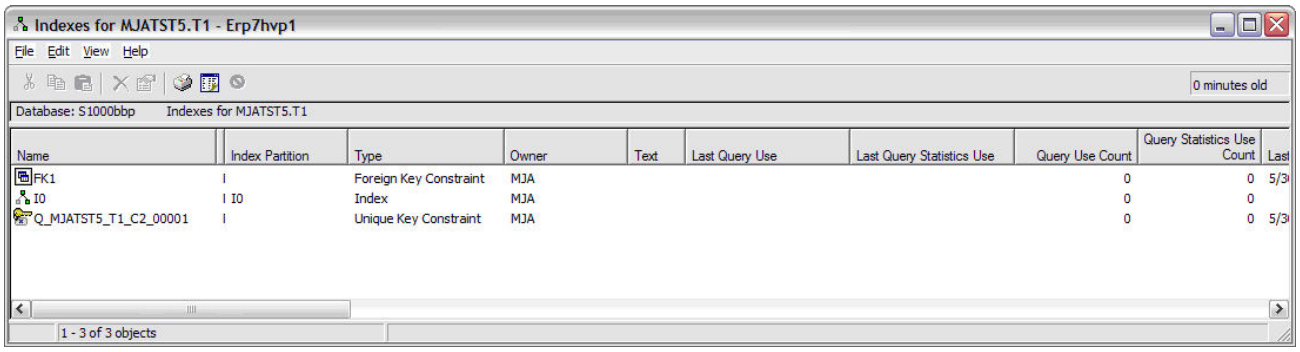


Figure 2. Show Indexes example



Show Indexes also only shows real indexes and not files that just share another file's index. Once you create your new index, refresh the Show Indexes list. If you see it in the Show Indexes list,

then one of the attributes must not be compatible. Compare the attributes of the new index against the attributes of the index you want to share to see what does not match. Correct the attribute in the CREATE INDEX statement or the DDS of the logical file and try again. The Show Indexes function also displays the media preference of each index so once you have created the sharing index and changed the media preference to SSD, refreshing the Show Indexes list will then show the SSD media preference.

CHGPFM and CHGLFM concurrency

As mentioned earlier, all previous mechanisms of changing the media preference required an exclusive lock on the entire file. This prevented changing the media preference if any application was currently accessing the file or any member of the file. Conversely, if an exclusive lock was able to be acquired, concurrent applications would not be able to access the file and could fail with a lock time-out.

CHGLFM and CHGPFM do not require an exclusive lock like the previous methods. A *SHRUPD lock is required, but this only conflicts with another job that has an *SHRNUP, *EXCLRD, or *EXCL lock on the file. At this time, an exclusive seize is acquired by the DB2 engine to synchronously move the data, so while a lock time-out will not occur, other jobs will wait for the move to complete. At some point we may change the commands to asynchronously move the data which would not cause other jobs to wait for the move to complete.

How do you know how much of an index or data is actually on SSD?

When you move an object on SSD, if insufficient space exists on SSD, only part of the object will end up on SSD. Because of this, you may have times when you may wonder how much of an object is actually allocated on SSD. Luckily, another set of enhancements were delivered with the following PTFs to address this issue.

- V5R4 (SF99504 Version 28)
- IBM i 6.1 (SF99601 Version 16)
- IBM i 7.1 (SF99701 Version 4)

New catalog views were provided to return information about the actual disk allocations for indexes and data.

- QSYS2.SYSPARTITIONDISK is a view that will return allocation information for DB2 for i tables and physical files. For example, the following Select statement will report on the data allocation for all SQL tables or physical files in schema (library) MJATST:

```
SELECT MAX(table_schema) AS table_schema, MAX(table_name) AS table_name,
   MAX(table_partition) AS table_partition,
   SUM(CASE WHEN unit_type = 1 THEN unit_space_used ELSE null END) AS ssd_space,
   SUM(CASE WHEN unit_type = 0 THEN unit_space_used ELSE null END) AS non_ssd_space
FROM qsys2.syspartitiondisk a
WHERE system_table_schema = 'MJATST'
GROUP BY a.table_schema, a.table_name, table_partition
ORDER BY 1,2,3
```

The following SELECT statement will return any SQL tables or physical files that have some allocation on SSD in schema (library) MJATST.

```
SELECT  DISTINCT table_schema, table_name, table_partition
FROM qsys2.syspartitiondisk a
WHERE system_table_schema = 'MJATST' and UNIT_TYPE = 1
```

- QSYS2.SYSPARTITIONINDEXDISK is a view that will return allocation information for DB2 for i indexes (i.e. keyed files, constraint indexes, and SQL indexes). For example, the following Select statement will report on the index allocation for all indexes on tables or physical files in schema (library) MJATST:

```
SELECT index_schema, index_name, index_member, index_type,
  SUM(CASE unit_type WHEN 1 THEN unit_space_used ELSE 0 END) AS ssd_space,
  SUM(CASE unit_type WHEN 0 THEN unit_space_used ELSE 0 END) AS nonssd_space
FROM qsys2.syspartitionindexdisk  b
WHERE system_table_schema = 'MJATST'
GROUP BY index_schema, index_name, index_member, index_type
order by 5 desc, 6 desc
```

Note: If you have a spanning index then you would need to divide by max (partnbr).

```
SELECT index_schema, index_name, index_member, index_type,
  SUM(CASE unit_type WHEN 1 THEN unit_space_used ELSE 0 END)/max(partnbr) AS ssd_space,
  SUM(CASE unit_type WHEN 0 THEN unit_space_used ELSE 0 END)/max(partnbr) AS nonssd_space
FROM qsys2.syspartitionindexdisk  b
WHERE system_table_schema = 'MJATST'
GROUP BY index_schema, index_name, index_member, index_type
order by 5 desc, 6 desc
```

The following SELECT statement will return any indexes on tables or physical files that have some allocation on SSD in schema (library) MJATST:

```
SELECT DISTINCT index_schema, index_name, index_member, index_type
FROM qsys2.syspartitionindexdisk  b
WHERE system_table_schema = 'MJATST' and UNIT_TYPE = 1;
```

Summary

CHGPFM and CHGLFM provide an easy and flexible way to move indexes and data on or off of SSD so you can make the most of your SSD investment. The enhanced commands do not require an exclusive lock like previous methods.

Resources

Additional information related to SSDs and recent enhancements can be found at the following:

- See How to boost application performance using Solid State Disk devices for more information about performance using Solid State Disks.
- For information on other mid-release technology updates, see IBM i Technology Updates.

- DB2 for i developerWorks forum

© Copyright IBM Corporation 2011
(www.ibm.com/legal/copytrade.shtml)
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)