Ergin Babani – ebabani@ca.ibm.com

02 April 2010

Moving tables online using ADMIN_MOVE_TABLE

Trademarks and Disclaimer

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS AND/OR SOFTWARE.

The information on the new product is intended to outline our general product direction and it should not be relied on in making a purchasing decision. The information on the new product is for informational purposes only and may not be incorporated into any contract. The information on the new product is not a commitment, promise, or legal obligation to deliver any material, code or functionality. The development, release, and timing of any features or functionality described for our products remains at our sole discretion.

Motivation

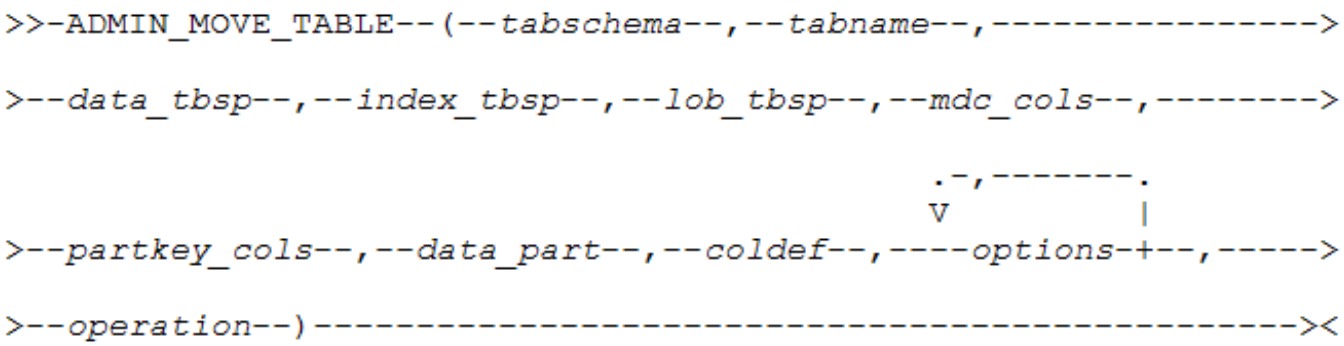
- Consider the following scenarios:
  - Customer wants to move their tables from a System Managed Space tablespace to a Database Managed Space tablespace
  - Customer wants to change the index or lob tablespace for a table
- Those operations require taking the tables down while the operation is taking place. Usually performed during planned maintenance while the system has been taken down.
- What if the operation would take longer than the maintenance window permits?

ADMIN_MOVE_TABLE

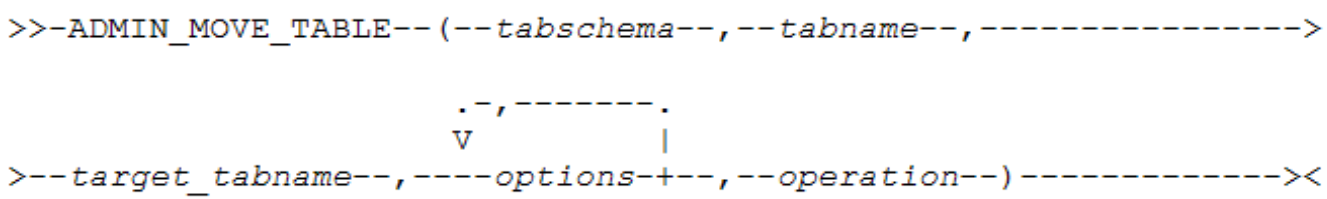
- Introduced in DB2 V9.7
- The ADMIN_MOVE_TABLE stored procedure moves the data in an active table to a new table object with the same name, while the data remains available for access.
- http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.sql.rtn.doc/doc/r0055069.html

ADMIN_MOVE_TABLE

- 2 Methods to call ADMIN_MOVE_TABLE
- 1



- 2



ADMIN_MOVE_TABLE

- There are three tables required for an online table move:
- **Source table**
  The original table name passed in to the procedure. This is the table to be moved.
- **Target table**
  The table where the data from the source table will be moved. This is either created by the procedure (if called using method 1) or the name of a table that already exists is passed on to the procedure (if called using method 2). At the end of the move this table will be renamed the same as the source table.

  During the move operation, we are still able to INSERT, DELETE or UPDATE rows on the source table. We need to keep track of these changes and update the target table accordingly. To help with this ADMIN_MOVE_TABLE creates a third table
- **Staging table**
  Records any INSERT, DELETE or UPDATE changes made to the source table while the procedure is running. The same changes are later replayed on the target table. This table is dropped at the end of the move.

ADMIN_MOVE_TABLE

- The ADMIN_MOVE_TABLE operation can be divided in 5 stages.

ADMIN_MOVE_TABLE – INIT Stage

INIT

- Actions performed during this stage:

1. Verify a table move can take place. Verify caller is authorized to perform the move.
2. If the caller passed in a target table by name, verify the target table has a valid table definition. If we have to create a target table, create it now.
3. Create a staging table.
4. Create 4 triggers. These triggers will capture any INSERT, DELETE and UPDATE changes on the source table, and record the changes on the staging table. If it's an UPDATE operation we will capture the value of the row BEFORE and AFTER the UPDATE.

ADMIN_MOVE_TABLE – INIT Stage

INIT

- To avoid any naming conflicts the utility will use the following method to create these temporary objects:
  <characters from name of object><base64 encoded hash key over name of object><postfix>
- Postfix:
  "t" for target
  "s" for staging
  "i" for insert trigger
  "d" for delete trigger
  "u" for before update trigger
  "v" for after update trigger
- Example:
  Assuming the source table is named T1:
  Staging table: T1AAAAVxs
  Target table: T1AAAAVxt
  Insert trigger: T1AAAAVxi
  Delete trigger: T1AAAAVxd
  Before Update trigger: T1AAAAVxu
  After Update trigger: T1AAAAVxv

ADMIN_MOVE_TABLE – COPY Stage

COPY

- Actions performed during this stage:

1. Copy the data from the source table to the target table.
2. Allow access to the source table while the COPY is taking place.
3. We have two ways to copy the data:
   1. Select data from source table using CS isolation level. Insert the data into the target table. Default.
   2. Create a cursor to select data from the source table. Load data from the cursor to the target table using a NONRECOVERABLE LOAD.
4. At the end of the COPY phase on the target table we create the same indexes as on the source table.

ADMIN_MOVE_TABLE – REPLAY Stage

REPLAY

- Actions performed during this stage:

1. During the COPY phase, record any changes to the source table in the staging table as well.
2. During the REPLAY phase, we go through the staging table and replay these same changes to the target table.
3. It is possible that rows were changing on the source table while we were performing the replay. After the replay, the staging table might still have data on it. We keep replaying until we reach a certain number of maximum replays(100), or until the staging table has 100 rows or less.

ADMIN_MOVE_TABLE – SWAP Stage

SWAP

- Actions performed during this stage:

1. During this stage we SWAP the name of the target table to that of the source table. The original source table is then removed.
2. Before we can swap the tables, we need to make sure that the source table and the target table have the same data. At this point the staging table might still have some data which has not been replayed.
3. We request an eXclusive lock on both the source table and the target table. Once we have the lock, we perform a final replay to make sure the source and target table have the same data. Since we have the eXclusive lock at this point, no new changes should have been made to the source table.
4. Remove the source table. Either drop the table, or if the KEEP option was specified rename it.
5. Rename the target table to the source table. Release the eXclusive lock.

ADMIN_MOVE_TABLE – CLEANUP Stage

CLEANUP

- Actions performed during this stage:

1. We still have a number of temporary objects left over from the run of the procedure. We need to remove these objects since they are no longer needed.
2. Delete all triggers that were created on the source table.
3. Delete the staging table.

ADMIN_MOVE_TABLE

- There are two ways to call ADMIN_MOVE_TABLE procedure

1. In a single step:
   ADMIN_MOVE_TABLE('EBABANI','T1','T1_TARGET','','MOVE')
   Perform every step of the operation.
2. Manually in multiple steps:
   ADMIN_MOVE_TABLE('EBABANI','T1','T1_TARGET','','INIT') // Perform INIT stage only
   ADMIN_MOVE_TABLE('EBABANI','T1','T1_TARGET','','COPY') // COPY stage. Can only be called after INIT
   ADMIN_MOVE_TABLE('EBABANI','T1','T1_TARGET','','REPLAY') // REPLAY STAGE. Can only be called after COPY.
   ADMIN_MOVE_TABLE('EBABANI','T1','T1_TARGET','','SWAP') // SWAP stage. Can be called after COPY or REPLAY stage.

ADMIN_MOVE_TABLE – Examples

- table T1(I1 int, I2 int)

TABSCHEMA	TABNAME	TBSPACE
EBABANI	T1	USERSPACE1

- Call ADMIN_MOVE_TABLE

```
hotel27:/home/ebabani> db2 "call SYSPROC.ADMIN_MOVE_TABLE('EBABANI','T1','USERSPACE2','USERSPACE2','USERSPACE2','','','','','','MOVE')"


  Result set 1
  --------------

  KEY                             VALUE
  ------------------------------- ----------------------------------------------------------------------------------------------------
  AUTHID                          EBABANI
  CLEANUP_END                     2010-04-02-09.13.49.023631
  CLEANUP_START                   2010-04-02-09.13.48.772233
  COPY_END                        2010-04-02-09.13.45.390419
  COPY_OPTS                       ARRAY_INSERT,NON_CLUSTER
  COPY_START                      2010-04-02-09.13.44.257124
  COPY_TOTAL_ROWS                 0
  INDEXNAME
  INDEXSCHEMA
  INIT_END                        2010-04-02-09.13.43.990619
  INIT_START                      2010-04-02-09.13.40.586361
  REPLAY_END                      2010-04-02-09.13.48.231429
  REPLAY_START                    2010-04-02-09.13.45.406674
  REPLAY_TOTAL_ROWS               0
  REPLAY_TOTAL_TIME               1
  STATUS                          COMPLETE
  SWAP_END                        2010-04-02-09.13.48.510543
  SWAP_RETRIES                    0
  SWAP_START                      2010-04-02-09.13.48.325362
  VERSION                         09.07.0002

  20 record(s) selected.

  Return Status = 0
hotel27:/home/ebabani>
```

ADMIN_MOVE_TABLE – Examples (cont.)

- table T1(I1 int, I2 int)

TABSCHEMA	TABNAME	TBSPACE
EBABANI	T1	USERSPACE2

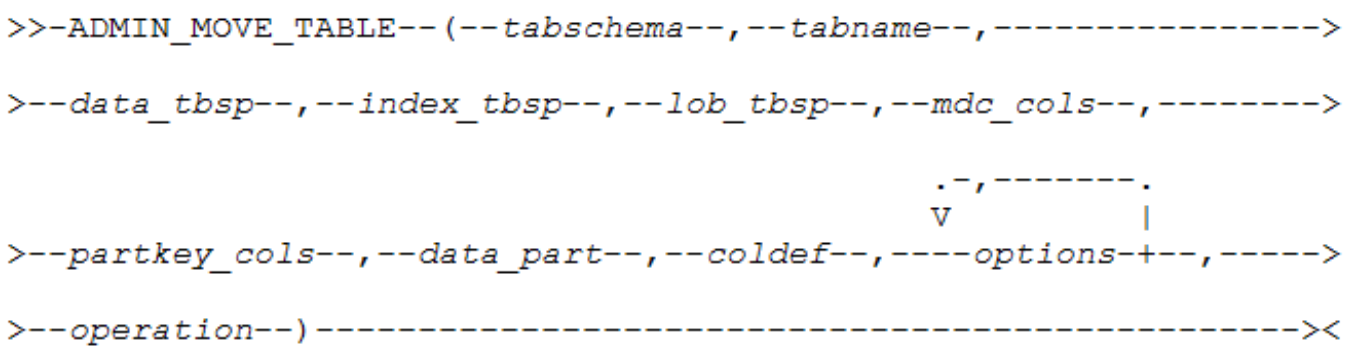
- Using db2look to verify
  db2look -d testdb -tw T1 -e

```
-------------------------------------------------
-- DDL Statements for table "EBABANI "."T1"
-------------------------------------------------


CREATE TABLE "EBABANI "."T1"  (
                  "I1" INTEGER ,
                  "I2" INTEGER )
                 IN "USERSPACE2" ;
```

ADMIN_MOVE_TABLE – Changing the table definition

- ADMIN_MOVE_TABLE can change the table definition during a move.



ADMIN_MOVE_TABLE - Example

- Changing table definition: Regular to MDC table
- db2 "call SYSPROC.ADMIN_MOVE_TABLE('EBABANI','T1','USERSPACE2','USERSPACE2','USERSPACE2','I2','','','','','MOVE')“

```
------------------------------------------------
-- DDL Statements for table "EBABANI "."T1"
------------------------------------------------


CREATE TABLE "EBABANI "."T1"  (
                  "I1" INTEGER ,
                  "I2" INTEGER )
                 IN "USERSPACE2"
                 ORGANIZE BY (
                  ( "I2" ) )
                 ;
```

- Table is now an MDC table after the move

ADMIN_MOVE_TABLE - Example

- Changing table definition: MDC to Regular table
- Can’t keep mdc_cols field empty. It will reuse the current MDC column specs in that case.
- Solution: Create the target table before calling the procedure.
- create table T2(I1 int, I2 int) in USERSPACE2
- db2 "call SYSPROC.ADMIN_MOVE_TABLE('EBABANI','T1',T2’,’’,’MOVE’)

```
------------------------------------------------
-- DDL Statements for table "EBABANI "."T1"
------------------------------------------------


CREATE TABLE "EBABANI "."T1"  (
                  "I1" INTEGER ,
                  "I2" INTEGER )
                 IN "USERSPACE2" ;
```

- Table is now a regular table again.

ADMIN_MOVE_TABLE - Example

- Changing column definition
- Table before move: T1 (C1 char(10), I2 SMALLINT)

```
------------------------------------------------
-- DDL Statements for table "EBABANI "."T1"
------------------------------------------------


CREATE TABLE "EBABANI "."T1"  (
                  "C1" CHAR(10) ,
                  "I2" SMALLINT )
                 IN "USERSPACE2" ;
```

SYSPROC.ADMIN_MOVE_TABLE('EBABANI','T1','USERSPACE2','USERSPACE2','USERSPACE2','','','','C1 char(100), I2 BIGINT','','MOVE')"

- Table after move: T1 (C1 char(100), I2 BIGINT)

```
------------------------------------------------
-- DDL Statements for table "EBABANI "."T1"
------------------------------------------------


CREATE TABLE "EBABANI "."T1"  (
                  "C1" CHAR(100) ,
                  "I2" BIGINT )
                 IN "USERSPACE2" ;
```