Avoiding Database REORG Operations in SAP Systems on DB2 for Linux, UNIX, and Windows Version 10.1

Applies to:

SAP NetWeaver 7.0 and higher on DB2 for Linux, UNIX, and Windows (DB2 for LUW) version 10.1 or higher (in the following referred to as DB2 10.1).

Summary

Database tables with frequent INSERT, UPDATE and DELETE operations become fragmented over time. This fragmentation causes problems in terms of performance and space reclamation. To alleviate these problems the database administrator starts a table and/or index reorganization which is a heavy operation and has an impact on the availability of that table or index.

DB2 10.1 offers new features that help to avoid reorganizations. This document discusses the usage of insert time clustering tables, index space reclamation and smart data and index prefetching in SAP systems running on DB2 10.1.

Author: Johannes Heinrich

Company: SAP AG

Created on: 01 October 2012

Author Bio

Johannes Heinrich works as a senior developer in the IBM DB2 for Linux, UNIX, and Windows development team at SAP. He holds a degree in Computer Science from the University of Kaiserslautern, Germany, and is an IBM Certified DBA for DB2 9 on Linux, UNIX, and Windows.

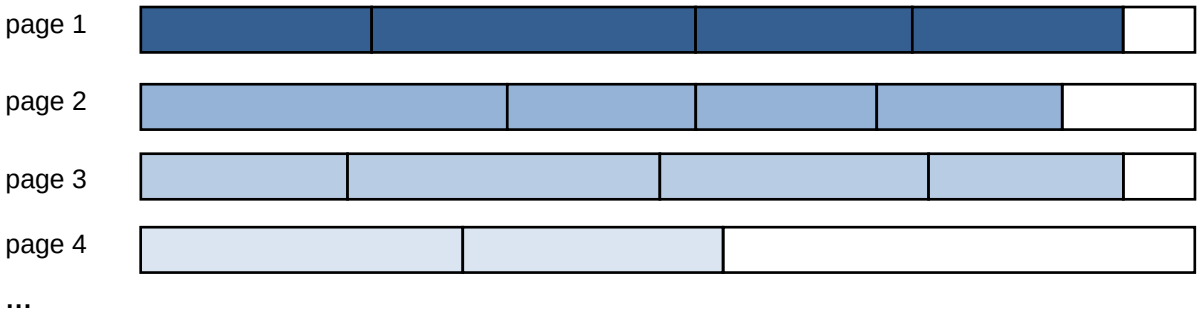
He can be reached at johannes.heinrich@sap.com.

Table of Contents

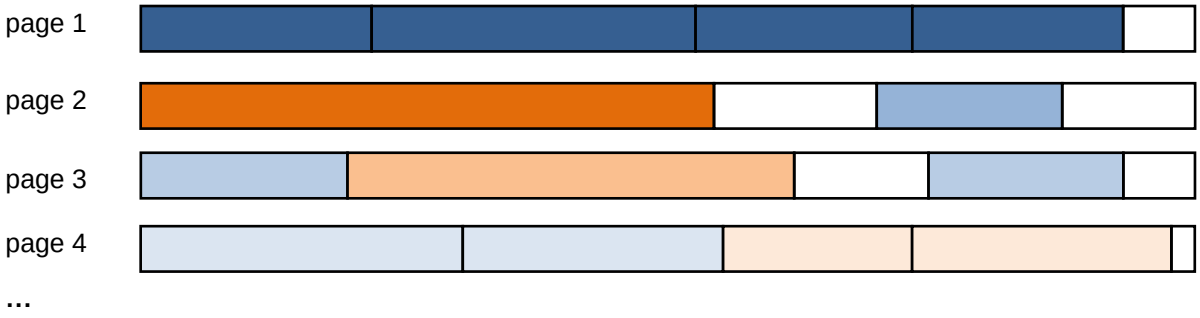
The Need for Reorganizations

In order to understand why reorganizations are important we need to take a look at how records are physically stored on disk. Here is a simplified description:

The database manager stores table data in so-called pages on the hard disk. A page constitutes the smallest piece of data that can be read by the database manager from the disk. DB2 lets you define the size of a page on disk during the database creation and while creating new tablespaces. In a standard SAP installation, each page has a size of 16 KB. When records are inserted into a table, the data pages are filled one by one in a *sequential, consecutive* order. If a record does not fit into one page it is inserted into the next page. This means that data that is inserted around the same time is also physically stored together. We say that data is clustered according to the insert time. This is shown in the following (simplified) figure:



Over time, existing table data is updated or deleted by the application. If, for example, records are deleted, the empty space will be reused by forthcoming inserts in this table as long as they will fit into the gap. Another example would be an update to a record so that it does not fit into its original page and must be inserted in another page (this is called an “overflow record”). All this means that over time the sequential order in which the records were inserted will be lost. Also, more and more unused space will appear in the pages:



This causes problems because

- The sequential prefetching algorithms used up to DB2 9.7 usually rely on the data clustering to be effective. Therefore, an unclustered table leads to *performance degradations*.
- Fragmented table data can lead to performance degradations also because more pages must be read to fetch all data for a given SELECT statement.
- Free space in the pages can only be reused by new records which are inserted into this table, not by new records inserted into other tables. In order to be used by other tables in the tablespace, the free space must be given back to the tablespace in an effective way without affecting ongoing operations. This process is also called *space reclamation*.

These problems can be addressed by a table reorganization. A reorganization optionally re-establishes the data clustering (if an appropriate index is used for the reorganization) and eliminates free space trapped in the data pages of the table.

Similar to tables, all indexes of a table are physically stored in one index object. Over time, the pages within this index object will be fragmented and sparsely filled so that a reorganization of the index becomes necessary.

Furthermore, data or index reorganizations are sometimes necessary to enable certain database functionality. Examples are the enablement of large RIDs after upgrading from an older DB release to DB2 9.1 or using classic row compression or index compression for a table which is not yet compressed. To summarize, reorganizations are necessary for the following reasons:

- To improve performance
- To reclaim trapped space within the physical database object
- To enable certain database features

Identifying candidates for reorganization and actually reorganizing tables and indexes is a DBA task and typically requires manual interactions with the database system (see below for automatic table reorganizations in DB2). Usually it also affects the availability of the table or index to the applications and requires disk and CPU resources from the DBMS. Consequently, you want to avoid reorganizations as much as possible.

Note: You can find a detailed discussion about when to perform reorganizations in a SAP system running on DB2 for LUW in SAP Note 975352 – I highly recommend reading this note.

Classification of Reorganization Methods

Several reorganization methods for tables and indexes are available with DB2 for Linux, UNIX, and Windows.

- The classic *offline REORG* (REORG TABLE...) basically works by creating a copy of the table that should be reorganized, replacing the old table with the new version and then recreating all indexes on the table. An advantage of this method is that the table data is afterwards perfectly clustered. On the other hand, the table is not accessible during the reorganization (only read access during some phases is possible) and additional disk space is required for the copy of the table. The offline REORG is also available for indexes (REORG INDEXES ALL or REORG INDEX <index name>).
- The *online REORG* (REORG TABLE...INPLACE) processes the table incrementally and therefore does not require much disk space. Also, with the exception of one phase, the table remains fully accessible. The online REORG moves data within the table object to minimize fragmentation. Disadvantages of this method include an imperfect data clustering upon completion and a potential high logging workload. The online REORG also needs to acquire different database locks during its operation that possibly slow down applications and the REORG operation itself.
  Indexes can be reorganized online by specifying …ALLOW WRITE ACCESS.
- You can also reorganize a table and re-create its indexes by using the *ADMIN_MOVE_TABLE* stored procedure to move the table and its indexes online within the same tablespace. In an SAP system you typically do so by running the report DB6CONV (see SAP Note 1513862). The ADMIN_MOVE_TABLE procedure copies the records of the existing table to a new table in the current order, which means that additional disk space is required. With the COPY_WITH_INDEXES option you can determine if the indexes for the target table are created before the records are copied or afterwards (which is the default).

Besides these reorganization techniques there are maintenance operations available that are also initiated with the REORG command. These operations are not reorganizations in the sense that data is shifted or copied. Instead, they only include updates on internal storage structures to release space from deleted records.

- For MDC tables, beginning with DB2 9.7 you can use the REORG command with the RECLAIM EXTENTS ONLY option for the purpose of space reclamation. MDC tables are always clustered per definition. See the SDN article Reclaiming Space from MDC Tables – Enhancements for DB2 9.7 for further details.
- For indexes, pseudo deleted keys or pseudo deleted pages (see section “Index Space Reclamation” below for an explanation) can be released with the REORG INDEXES… CLEANUP ALL or

CLEANUP PAGES option. Released pages can be reused by the indexes of the respective table, not by indexes from other tables.

These maintenance operations work in an online fashion. They are characterized by a very low resource consumption compared to the reorganization methods mentioned above, and they are usually fast. DB2 10.1 offers two new maintenance operations, namely the reclamation of extents for ITC tables and the index space reclamation. Both methods are discussed in detail in the following sections.

Insert Time Clustering Tables

ITC tables, a new table type available as of DB2 10.1, build on the infrastructure of MDC tables and try to combine their advantages with regard to clustering and easy, light weight space reclamation with the ease of use of standard tables. The effectiveness of the space reclamation for ITC tables depends on the access pattern.

Note: ITC tables offer a good space reclamation method if *data that was inserted together will be also deleted together*.

In an SAP system, we typically see this access pattern for tables that can be archived, especially if archiving is based on a time-related criterion (for example, “archive all billing documents that were created before 2010”). On the other hand, it does not make sense to use ITC tables for the purpose of easier space reclamation if deletions appear to be random and not ordered by time.

Technical Considerations

ITC tables are created by specifying the keyword “ORGANIZE BY INSERT TIME” in the CREATE TABLE statement. ITC tables cluster regular data based on the insert time of the record – data which is inserted around the same time is placed close together. You can think of an ITC table as an MDC table that uses a virtual, non-existing field that contains the insert time as the only dimension. If MDC tables are new for you, read the document Multi Dimensional Clustering and Deployment in SAP Business Warehouse as an introduction.

ITC tables, like MDC tables, are organized in blocks. Each block consists of a number of pages as determined by the extent size. In a typical SAP system, we use an extent size of 2. Given the standard 16K pages, a block consists of 32K. The rather small extent- and block size is an advantage – smaller blocks tend to be more often completely empty than larger blocks, and as of DB2 10.1 only completely empty blocks can be reclaimed.

MDC and ITC tables use a special type of indexes, so called block indexes. In the MDC case, block indexes can help to retrieve data very efficiently. In the case of ITC, for technical reasons one block index is created automatically by the database manager. It is based on a virtual column (the insert time) and because this column does not exist in the table, it cannot appear in queries and this block index will not be chosen by the optimizer.

To manage block allocation and quickly find empty blocks, DB2 creates a so-called block map. The block map is stored as a separate object and contains an entry for each block of the table which specifies its allocation status (free, in use etc.).

Refer to Table and index management for MDC and ITC tables in the IBM DB2 10.1 Information Center for a good description of the structure of MDC and ITC tables.

Using ITC Tables: An Example

In the following I will walk through an example of how ITC tables can be used in an SAP NetWeaver 7.31 system. The example is designed in a way so that you can reproduce it in your own ABAP based (test) system on SAP NetWeaver 7.0 or higher.

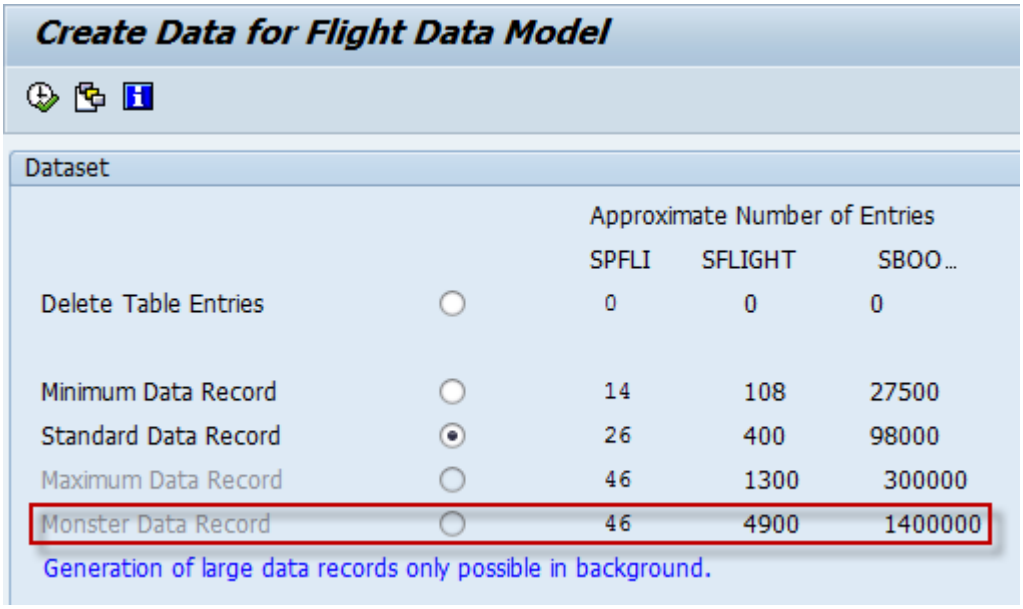
Note: To use ITC tables in an AS (Application Server) ABAP installation, *your system must have the SPs mentioned in SAP Note 1701181 (“DB6: ABAP DDIC: Enhancements for DB2 10.1”) or you must apply the correction instructions provided by this SAP Note*. These corrections ensure for example that you can convert existing non-ITC tables to ITC with DB6CONV. Furthermore, a DDIC conversion of an ITC table (e.g. during an EHP installation

or an SAP system upgrade) will preserve the ITC status of that table.
At the time of writing this article there are no such corrections available for the AS Java. Here, you can convert existing non-ITC tables to ITC tables as well, e.g. by using the ADMIN_MOVE_TABLE stored procedure directly, but during a Java DDIC conversion an ITC table will be reverted back to a regular table.

Creating Suitable Test Data

Standard SAP NetWeaver systems contain a report that creates data for the tables that are used in the famous ABAP flight demo example. We will use this report to create a considerable number of flight bookings (which translates to entries in the SBOOK table), transform the SBOOK table into an ITC table and start archiving records from it. Afterwards we study the effects of a REORG…RECLAIM EXTENTS operation on the space allocation.

Now start report SAPBC_DATA_GENERATOR.



We are going to create the “Monster Data Record”. To do so, schedule this report as a batch job with SAP transaction SM36 by using the predefined variant SAP&BC_MONSTER. Monitor the batch job with SAP transaction SM37 and wait until it finishes.

On the DB2 command line, issue the following SELECT statement (replace SAPMI1 with your schema name):

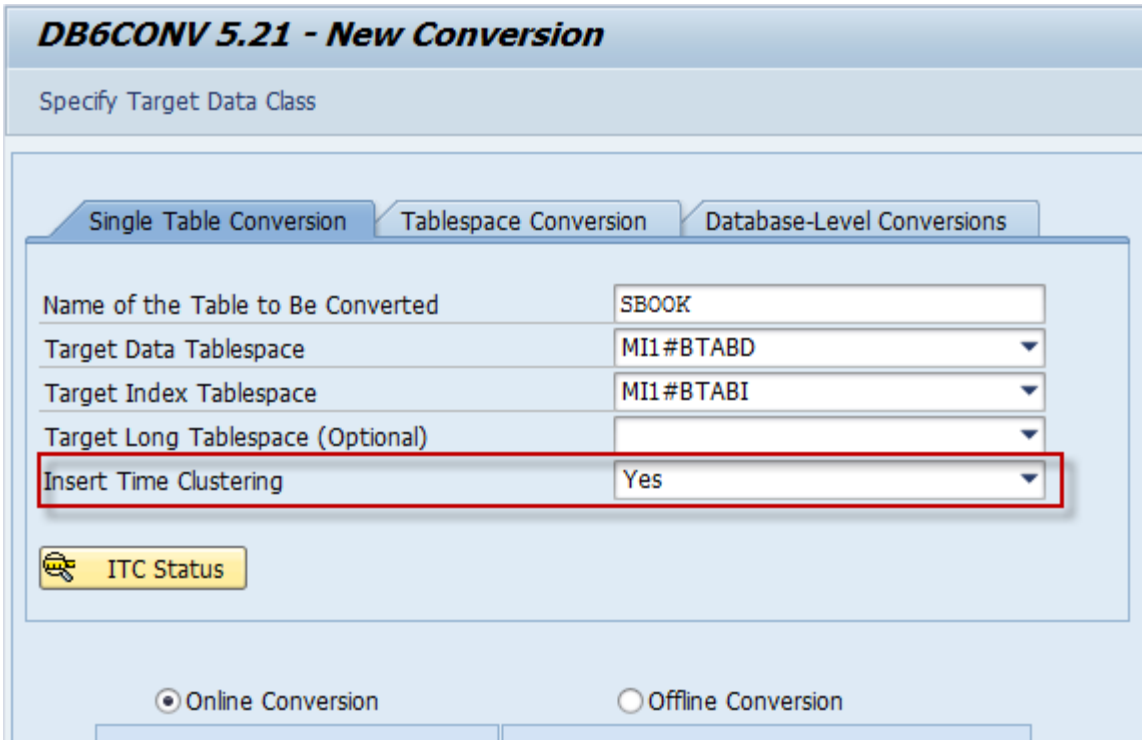
```
db2 "SELECT data_object_p_size, index_object_p_size, reclaimable_space FROM
TABLE(SYSPROC.ADMIN_GET_TAB_INFO('SAPMI1', 'SBOOK')) AS T"

DATA_OBJECT_P_SIZE   INDEX_OBJECT_P_SIZE  RECLAIMABLE_SPACE
-------------------- -------------------- --------------------
              236384                96992                    0
```

Table SBOOK needs about 230 MB disk space for its data and around 95 MB for its indexes. Not surprisingly, so far there is no space to reclaim.

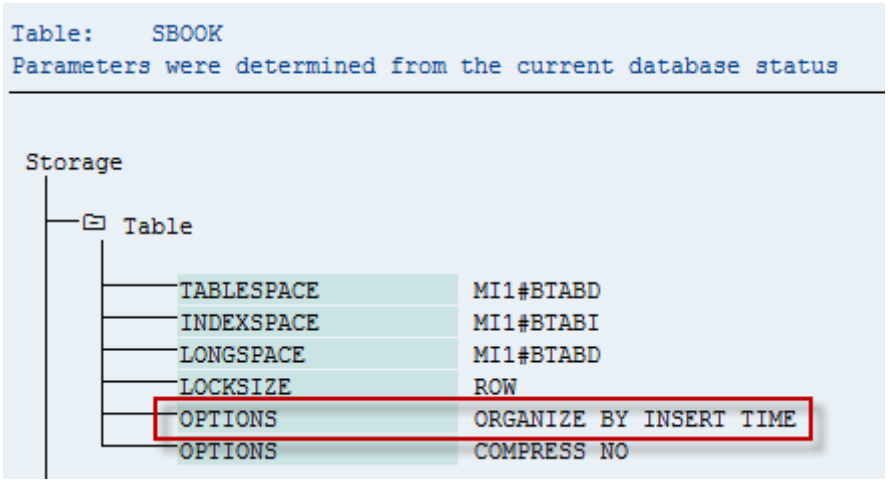
Converting to an ITC Table

Up to now we only inserted data into table SBOOK. Before we actually perform some deletions by means of data archiving we need to convert this standard table to an ITC table. In a system based on AS ABAP the best way to do this is to use report DB6CONV version 5.20 or higher (latest version recommended) and perform an online table move.

Start report DB6CONV and select “New Conversion”. On the “Single Table Conversion” tab, enter SBOOK as table name, enter the tablespaces that SBOOK is currently using and select “Yes” for “Insert Time Clustering”. Also make sure that you select “Online Conversion”.



Save the data, start the conversion and monitor it. After it finishes call SAP transaction SE14, enter SBOOK as table name and choose “Storage Parameters”. This should show that SBOOK is now an ITC table:



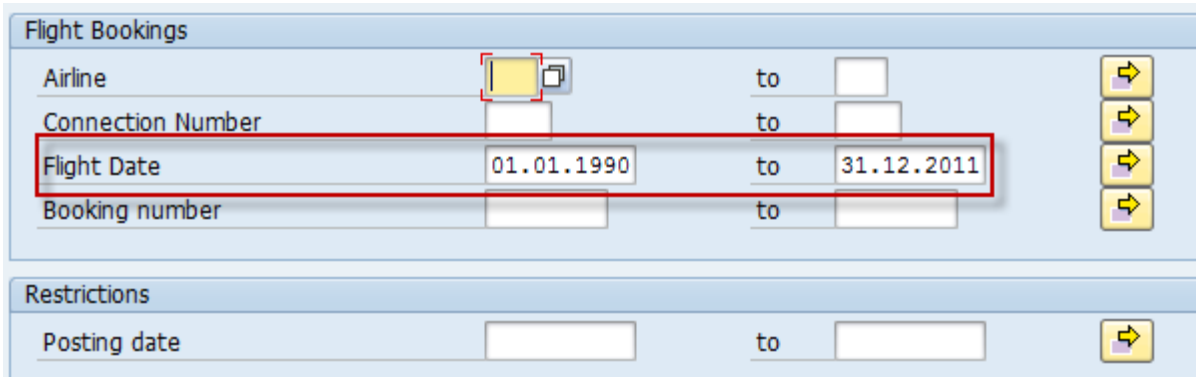
Report DB6CONV has also the ability to propose suitable candidates for ITC conversion based on the archiving statistics of your system. More on this later.

To analyze the effects of space reclamation it is useful to examine the block map mentioned above. You can do so by using the /DM action of the db2dart tool to generate a textual report that contains the block map of table SBOOK (note that running db2dart is an offline operation). Based on such a report I created the following figure which shows the current allocation of the first 600 blocks:

The first block is reserved for system usage; all other blocks are filled with data. No surprise here.

Archiving Data From SBOOK

The tables of the ABAP flight demo example have their own archiving object. To archive records out of table SBOOK and the other example tables, go to SAP transaction SARA and enter BC_SBOOK as archiving object. Assuming the basic archive customizing is in place, select the *Write* action and create a variant for the write program. Let's assume that we want to archive all flight bookings with a flight date older than 01.01.2012:



By doing so we archive based on a time related criterion. This improves our chances that we meet the required access pattern for insert time clustering tables (data that is inserted together should be deleted together) because the INSERTs were also based on a time related criterion (the insert time). Remember that we can only reclaim space from completely empty blocks. Also note that other archiving criteria are available here (such as connection number or airline) that might not be time-related. So the fact that we archive from a table does not necessarily imply that this table is a suitable candidate for ITC, but the selection criteria in the write program are also an important factor.

After specifying a start date and the spool parameters, run the archiving job and wait until it finishes. Then select the *Delete* action in SAP transaction SARA, choose the archive that was just created, specify start date and spool parameters and run the delete job.

After performing these actions, the table size has not changed, but now we see that there is some reclaimable space:

```
db2 "SELECT data_object_p_size, index_object_p_size, reclaimable_space FROM
TABLE(SYSPROC.ADMIN_GET_TAB_INFO('SAPMI1', 'SBOOK')) AS T"
```

```
DATA_OBJECT_P_SIZE   INDEX_OBJECT_P_SIZE  RECLAIMABLE_SPACE
-------------------- -------------------- --------------------
              236384                96992               100704
```

The interesting thing now is how the block map of table SBOOK has changed. This is the figure generated with the help of the db2dart tool after archiving took place:

The important point here is that there are groups of completely empty (“FREE”) blocks that can be reclaimed now. Remember that “empty block” here means that all records that are located on the two pages which constitute this block are deleted. The distribution of these blocks is not important and depends on the application (in our case the report which created the sample data). As you can see, our archiving criterion (flight date) does not perfectly match the insert order of this table, but this is no problem. No matter where the empty blocks are located, we can start reclaiming their space now.

Also note that the blocks marked as “FREE” here could be re-used for forthcoming INSERTs into this table. When inserting new records into an ITC table, DB2 ensures that data within a block is inserted at similar times. Adjacent blocks may not contain data that was inserted at a similar time.

Reclaim Space from ITC Tables

You perform the following command on the DB2 command line to reclaim unused storage in the table objects for table SBOOK:

```
db2 "REORG TABLE SAPMI1.SBOOK RECLAIM EXTENTS ALLOW WRITE ACCESS"
```

Note: DB2 9.7 uses RECLAIM EXTENTS **ONLY**, while DB2 10.1 has RECLAIM EXTENTS without the ONLY keyword in the REORG command syntax. However, REORG… RECLAIM EXTENTS ONLY works also in DB2 10.1.

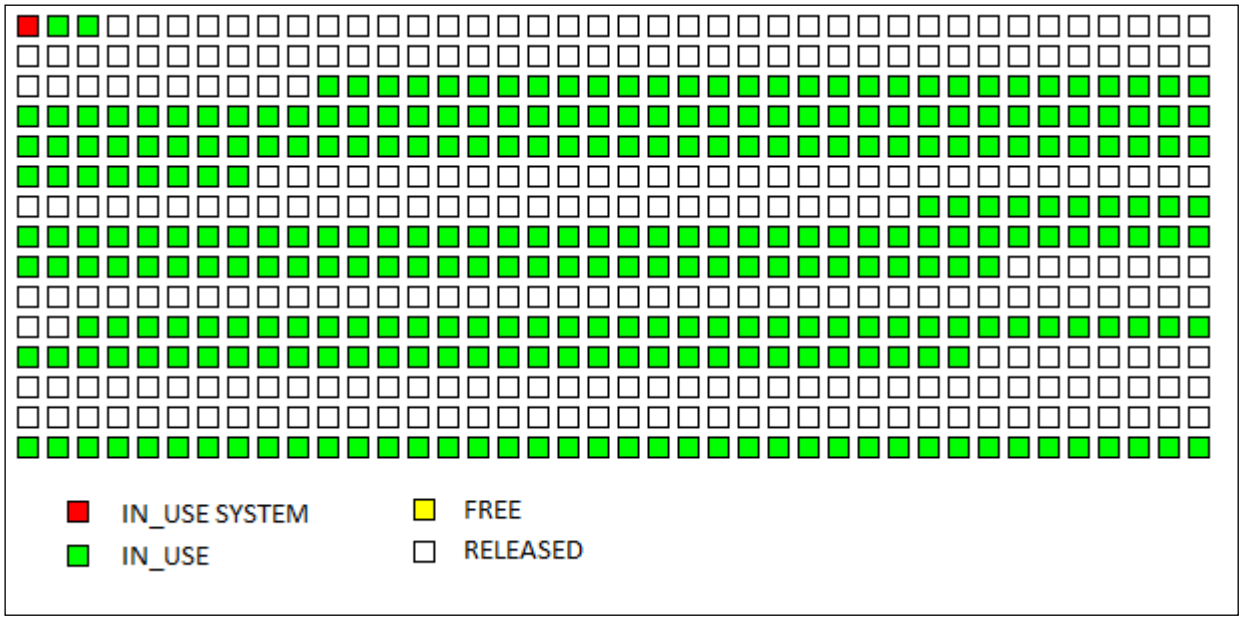
Reclaiming extents is very fast and it can be performed online, with read access or offline. See the SDN article Reclaiming Space from MDC Tables – Enhancements for DB2 9.7 for an explanation of these modes and their locking behavior.

As expected, the above statement reclaims all empty blocks and therefore shrinks the size of the table:

```
db2 "SELECT data_object_p_size, index_object_p_size, reclaimable_space FROM
TABLE(SYSPROC.ADMIN_GET_TAB_INFO('SAPMI1', 'SBOOK')) AS T"

DATA_OBJECT_P_SIZE   INDEX_OBJECT_P_SIZE  RECLAIMABLE_SPACE
```

```
-------------------- -------------------- --------------------
              135680                96992                    0
```

Blocks that were marked as “FREE” in the block map are now in status “RELEASED” which means that they are given back to the tablespace; they can now be acquired by any table in the tablespace. The block map itself does not shrink; existing entries in the block map with status “RELEASED” can be reused later.



The figure also shows that the REORG…RECLAIM EXTENTS operation did not include any data movement.

Additional Considerations

Before using ITC tables in your production systems consider the following points:

- ITC tables must maintain the additional block index mentioned above. Compared to regular tables, write operations on ITC tables may have a slight performance overhead (less than 5 percent).
- You cannot reorganize ITC tables online (REORG…INPLACE) – However, this should not be necessary anyway.
- REORG operations, especially the online REORG with REORG…INPLACE, rearrange the records in the table. Tables that were often reorganized are therefore not good candidates for a conversion to an ITC table.
- If an updated record does not fit in its block anymore it is stored in a block with enough free space. This record then does not follow the insert time order anymore.
- You must not convert SAP NetWeaver BW tables to ITC. For BW, the use of MDC is preferred. Space reclamation for MDC tables works in the same way as for ITC tables. Additionally, MDC often leads to considerable query performance improvements in a BW environment.

Finding Suitable ITC Candidates

After an upgrade of an existing SAP system to DB2 10.1 you probably want to find out which regular tables are suitable candidates for the conversion to ITC tables. Candidates are

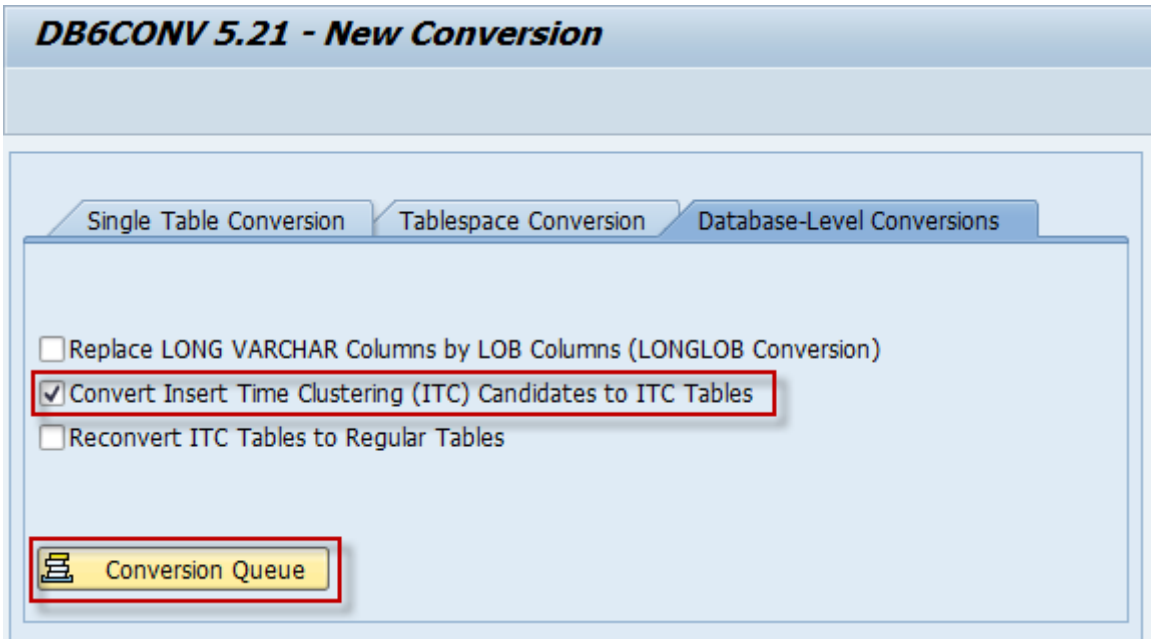
- Tables from which records were actually deleted
- Tables that show the typical access pattern (data that is inserted together is also deleted together).

In an SAP system, this usually applies to tables from which data is archived. Beginning with version 5.20, report DB6CONV is able to identify such candidates based on the statistics of the SAP Archive Development Kit (ADK). Upon request DB6CONV examines these statistics and present all tables

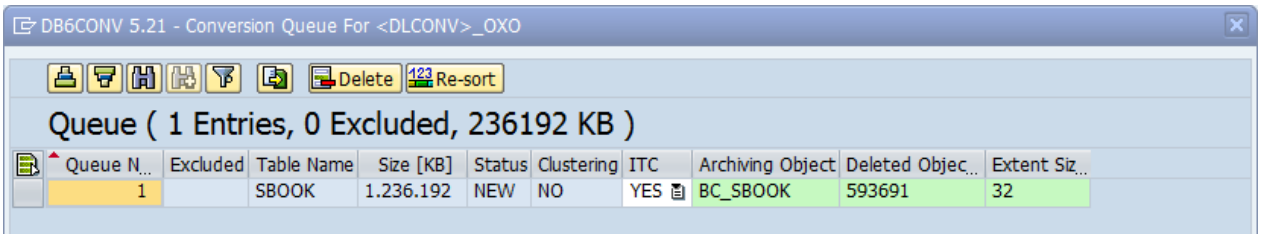
- That are not already ITC or MDC tables.
- That are at least 1 GB in size.

- That are part of an archiving object for which at least one completed archiving session exists. Usually this means that data was actually deleted from this table, although exceptions from this rule are possible.

You can access this functionality by selecting the *Convert Insert Time Clustering (ITC) Candidates to ITC Tables* checkbox on the *Database-Level Conversions* tab within the *New Conversion* screen and choosing *Conversion Queue*:



Assuming our SBOOK table in the example above is larger than 1 GB and has not already been converted to an ITC table, DB6CONV would display a dialog box like this:



This informs you that table SBOOK with a current size 1.2 GB that belongs to the archiving object BC_SBOOK was identified as a candidate for ITC conversion. 593.691 BC_SBOOK objects were archived and deleted. Note that this does not necessarily mean that the same number of records was deleted from table SBOOK. The table resides in a tablespace with an extent size of 32 KB. The extent size is an indicator for the likelihood of completely empty blocks after archiving sessions. Now it is your decision if you actually convert the table to ITC (set column *ITC* to *YES*) or not (set column *ITC* to *NO*).

Note: The fact that a table appears as candidate in DB6CONV is a strong indication that it is suitable for a conversion to ITC. However, the actual deletion pattern highly depends on the criteria used for archiving. Depending on these criteria it is possible that actually no or not enough records are deleted from the table under investigation and consequently no blocks will be completely empty and no disk space can be reclaimed.

Index Space Reclamation

Before DB2 10.1, to reclaim unused space in index objects you need to issue an index reorganization (e.g. REORG INDEXES ALL FOR TABLE <tabname>). This index reorganization usually requires a lot of log space for its long running transactions and temporarily additional disk space. With DB2 10.1, there is a new option available that releases unused space within index objects in an online fashion, avoiding the disadvantages mentioned before.

If records in a table are deleted, DB2 does not immediately delete the corresponding key entries in the indexes of this table[1]. Instead, these index entries are marked as “pseudo-deleted”. The REORG command together with the CLEANUP keyword can be used to remove these pseudo-deleted keys from the indexes, for example:

```
db2 "REORG INDEXES ALL FOR TABLE SAPMI1.SBOOK ALLOW WRITE ACCESS CLEANUP ALL"
```

This functionality has been available before DB2 10.1. It frees up space which can be re-used by all indexes that are defined on the corresponding table. Beginning with DB2 10.1, however, new functionality is available that allows returning the unused space in the indexes so that it can be used by other objects in this tablespace as well. This is called “Index Space Reclaim”.

Note: The index space reclaim functionality is available after upgrading to DB2 10.1. It is not necessary to rebuild existing indexes.
This functionality is also available for all types of indexes, independent from the related table type. ITC tables are not a requirement for using the index space reclaim.

The index space reclaim in DB2 10.1 moves *pages* within the index object to form empty *extents*. In a second step, these extents are freed so that they can be used by other database objects in the same tablespace. Like other REORG operations the index space reclaim operation can be carried out offline (ALLOW NO ACCESS) or online (ALLOW READ ACCESS, ALLOW WRITE ACCESS). Depending on the access mode and whether space from the indexes of a regular table or from a partition of a partitioned table is reclaimed different locks are required. Independent of the access method, at some point in time the index reclaim operation needs to wait for existing transactions on the table to finish while allowing new requests to access the table and its indexes.

To issue an index space reclaim you use the REORG INDEXES statement with the new RECLAIM EXTENTS keyword. Compared to an index reorganization which rebuilds the indexes (e.g. REORG INDEXES ALL FOR TABLE <tabname>), the new method requires less logging space, involves no long running transactions and does not require temporarily a significant amount of disk space.

Index Space Reclaim Example

Let us continue our example with the SBOOK table from above. During archiving, not only table data was deleted, index entries that refer to the deleted table data can also be removed. Consequently, it must be possible to release space from the indexes of table SBOOK, which are as follows:

```
db2 "SELECT CAST(indname AS VARCHAR(30)), indextype FROM syscat.indexes WHERE tabname
= 'SBOOK'"

1                              INDEXTYPE
------------------------------ ---------
SQL120911101725290             DIM
SBOOK~0                        REG
SBOOK~ACY                      REG
SBOOK~CUS                      REG
```

We have three regular indexes and one block index for the ITC table which is necessary as explained above. All indexes are stored into one physical index object. To figure out if there is some space that can be given back we can use the ADMIN_GET_INDEX_INFO function as in the following example:

```
db2 "SELECT CAST(indname AS VARCHAR(30)), index_object_p_size, reclaimable_space FROM
TABLE(sysproc.admin_get_index_info('T','SAPMI1','SBOOK')) AS T"

1                              INDEX_OBJECT_P_SIZE  RECLAIMABLE_SPACE
```

[1] If the currently active transaction holds an X lock on the table the corresponding index entries are actually deleted immediately.

```
------------------------------ -------------------- --------------------
SQL120911101725290                              96992                41184
SBOOK~0                                         96992                41184
SBOOK~ACY                                       96992                41184
SBOOK~CUS                                       96992                41184
```

The output is misleading, it does *not* mean that every index is 94 MB in size and 40 MB disk space can be reclaimed. Instead, as all indexes are located in one physical object, it means that all indexes together take up to 96 MB and 40 MB disk space can be reclaimed in overall.

For best results the cleanup of the pseudo-deleted keys and the index space reclaim should be combined like in the following statement:

```
db2 “REORG INDEXES ALL FOR TABLE SAPMI1.SBOOK ALLOW WRITE ACCESS CLEANUP ALL RECLAIM
EXTENTS”
```

We can check now either with ADMIN_GET_TAB_INFO or with ADMIN_GET_INDEX_INFO if the space was actually reclaimed:

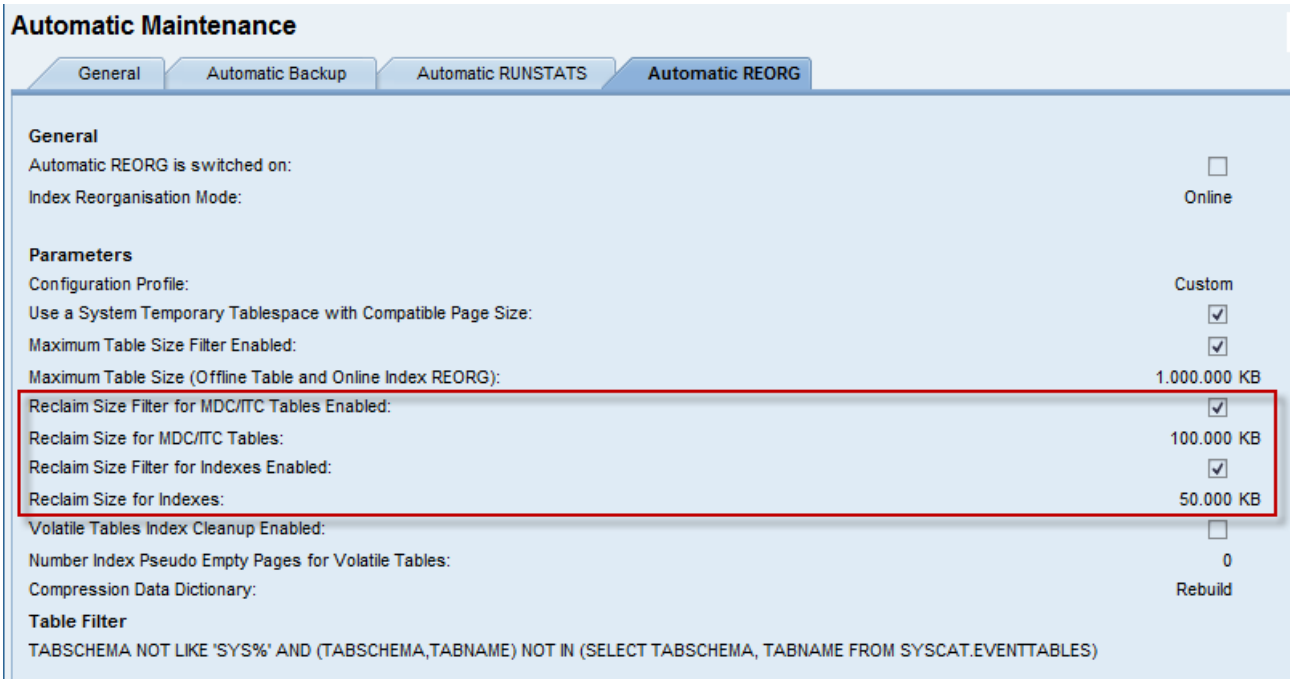
```
db2 "SELECT CAST(indname AS VARCHAR(30)), index_object_p_size, reclaimable_space FROM
TABLE(sysproc.admin_get_index_info('T','SAPMI1','SBOOK')) AS T"

1                              INDEX_OBJECT_P_SIZE  RECLAIMABLE_SPACE
------------------------------ -------------------- --------------------
SQL120911101725290                              55712                    0
SBOOK~0                                         55712                    0
SBOOK~ACY                                       55712                    0
SBOOK~CUS                                       55712                    0
```

The trapped space was reclaimed successfully; the physical object size of the index was actually reduced.

Automation Policy for Space Reclamation

To automate table and index maintenance tasks, including the identification of tables that are suitable for reorganization and the actual reorganization of these tables, DB2 offers so-called “autonomic features”. Based on an autonomic policy that is stored as an XML document in the database server the database manager performs reorganizations during a defined time period automatically.

You can add the new RECLAIM EXTENTS functionality for ITC tables and indexes to the automation policy of a DB2 10.1 database. To check the current settings you can use a current version of the WebDynpro-based DBA Cockpit (SAP NetWeaver 7.02 SP12, NW7.30 SP8, NW 7.31 SP5 or higher). Here, under *Configuration→Automatic Maintenance,* you can find the following settings on the *AUTOMATIC REORG* tab:



As an alternative, the following statement on the DB2 command line fetches the current version of the automatic reorganization policy into <policy_file>:

```
db2 "CALL SYSPROC.AUTOMAINT_GET_POLICYFILE('AUTO_REORG', '<policy_file>')"
```

If you run this statement as DB2 instance owner the <policy_file> is afterwards located in the home directory of the instance owner under *sqllib/tmp.* Here is an example of a reorg automation policy:

```
<?xml version="1.0" encoding="UTF-8"?>
<DB2AutoReorgPolicy xmlns="http://www.ibm.com/xmlns/prod/db2/autonomic/config">
      <ReorgOptions dictionaryOption="Rebuild" indexReorgMode="Online"
            useSystemTempTableSpace="true"
            reclaimExtentsSizeForTables="100000"
            reclaimExtentsSizeForIndexObjects="50000" />
      <ReorgTableScope maxOfflineReorgTableSize="1000000">
            <FilterClause>TABSCHEMA NOT LIKE 'SYS%' AND (TABSCHEMA,TABNAME) NOT IN
                  (SELECT TABSCHEMA, TABNAME FROM SYSCAT.EVENTTABLES)
            </FilterClause>
      </ReorgTableScope>
</DB2AutoReorgPolicy>
```

In this example, the *reclaimExtentsSizeForTables* attribute specifies that the RECLAIM EXTENTS operation runs automatically for MDC and ITC tables if 100 MB or more free space can be reclaimed. The reclaimExtentsSizeForMDC attribute which is available in DB2 9.7 is deprecated in DB2 10.1 and was replaced with this new reclaimExtenstsSizeForTable attribute that now considers also ITC tables. Similar to tables, the new *reclaimExtentsSizeForIndexObjects* attribute specifies that the RECLAIM EXTENTS operation for indexes runs automatically if 50 MB or more of trapped space can be reclaimed from the index object (that is, from all indexes of a table). Also note that the *maxOfflineReorgTableSize* element will not apply to the RECLAIM EXTENTS operation for MDC and ITC tables. For a table which is larger than *maxOfflineReorgTableSize* only index cleanup and reclaim operations will be considered by DB2. An index reorg which rebuilds the index will not be executed.

You can edit this file manually and store it again on the DB2 server with the help of the AUTOMAINT_SET_POLICY stored procedure.

Note: The current version of the db6_update_db script (see SAP Note 1365982) that must be executed after a database upgrade modifies the automatic reorg policy in a way so that RECLAIM EXTENTS operations for tables and indexes are carried out automatically (as shown in the example above).

For further details and recommendations regarding the automatic reorganization of tables in an SAP system running on DB2 for LUW see section 6 in the document IBM DB2 Autonomics - A Practical Guide to DB2 Autonomic Features in SAP Environments and SAP Note 975352.

Smart Prefetching

In the previous sections we discussed how to reclaim unused space without costly table and index reorganizations. We now turn our attention to the question how we can improve the performance of queries despite the fact that data and index pages are unclustered after many INSERT, UPDATE and DELETE operations.

Prefetching is a technique used by database systems to fetch pages from the disks into the buffer pool with the expectation that they are needed by the application. If they are actually needed, the database manager can access them in the buffer pool instead of issuing expensive direct read I/O operations. Prefetching is based on the concept of locality: if an application requires access to a specific record, there is a high probability that it will also access records that are stored close to the requested record.

We distinguish between

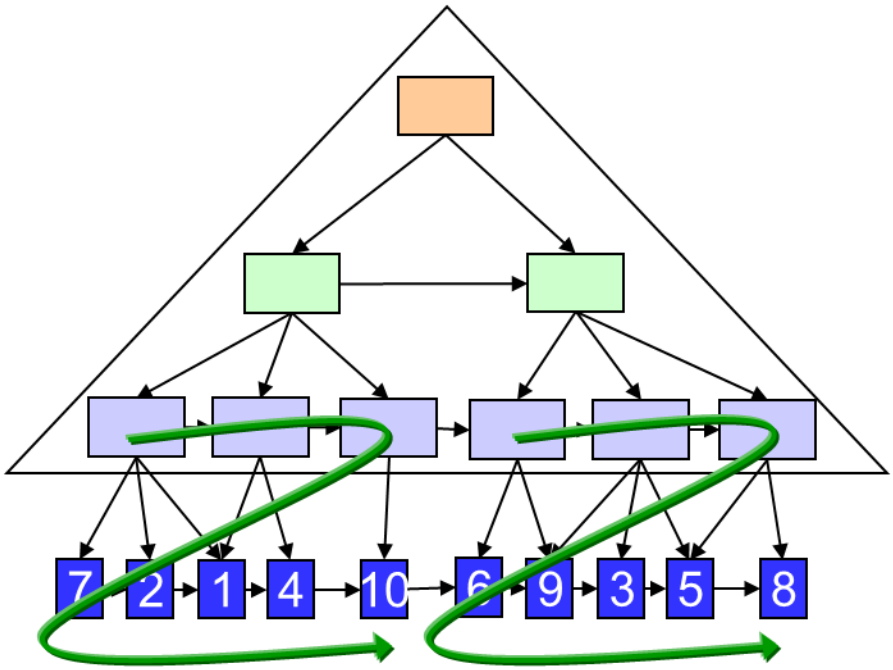
- Data prefetching, where data pages of tables are read in advance to satisfy forthcoming data read requests from within the buffer pools
- Index prefetching, where index pages are read in advance to satisfy forthcoming index read requests from within the buffer pools

Furthermore, we can categorize prefetching techniques in DB2 10.1 as follows:

- *Sequential prefetching,* which reads consecutive pages into the buffer pools.
- *Readahead prefetching* (new in DB2 10.1), which reads pages based on their order in the index into the buffer pools
- *List prefetching,* which prefetches a set of nonconsecutive pages efficiently.

Smart Data Prefetching

A very common access pattern in database systems is the scanning of an index to find the location of records that satisfy a specific query and then fetching these pages into the buffer pools. In DB2 this is also called an IXSCAN-FETCH operation. The performance of such an operation degrades if the data pages are unordered with respect to the scanned index due to many update, insert and delete operations because they cannot benefit from the sequential prefetching technique anymore. Sequential prefetching simply becomes ineffective. Furthermore, pages can be clustered only with regard to one index, which means that the data pages are always unclustered in relation to additional indexes. The following figure shows a (simplified) physical representation of an index, each box represents a page:



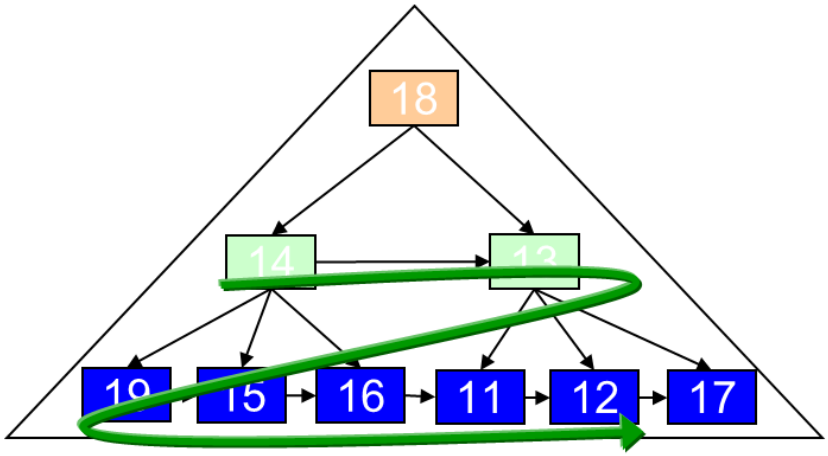
Clearly, the data pages at the bottom are unordered and therefore unclustered with regard to this index. As of DB2 10.1 the optimizer can enable readahead prefetching in such situations which means that the pages that are fetched during an IXSCAN-FETCH operation are determined by the leaf pages of the corresponding index as shown by the green line. The prefetchers then issue multiple data page I/O operations in one prefetch I/O operation to fetch exactly the needed pages. Since readahead prefetching requires additional resources to locate these pages, it is the task of the optimizer to choose the best prefetching method for a given query. The optimizer may decide to use one of the following options:

- Sequential prefetching.
- Readahead prefetching.
- Both sequential and readahead prefetching. In this case the query execution starts with sequential prefetching until a specific threshold of non-prefetched pages is reached; then, it switches to readahead prefetching. This is the default prefetching method.
- No prefetching.

Automatically deciding on the most efficient table data prefetching method is referred to as *smart data prefetching*.

Smart Index Prefetching

Similar to fetching data from regular tables, physical pages from index objects need to be fetched into the buffer pools during an IXSCAN operation. Again, prefetching techniques are used to speed up this process. Following many insert, update and delete operations the sequential prefetching of index pages can become ineffective because index leaf pages become non-sequential. Therefore, readahead prefetching of index pages is available beginning with DB2 10.1.This technique looks one level above the leaf pages of an index to identify exactly the pages a particular index scan needs. These pages are then prefetched as indicated by the green line in the following figure:



The figure shows the physical representation of an index, the numbers refer to page numbers of a physical index object.

Again additional resources are required to locate these pages, and therefore the optimizer decides about the best prefetching technique. Smart index prefetching uses the same prefetching methods as smart data prefetching (none, sequential, readahead, sequential and readahead). Automatically deciding on the most efficient index data prefetching method is referred to as *smart index prefetching*.

Smart Prefetching Configuration and Monitoring

Smart prefetching is enabled by default in DB2 10.1 via the database configuration parameter SEQDETECT that can be set to YES (enable smart prefetching) or NO (disable smart prefetching). Note that the meaning of the SEQDETECT configuration parameter has been changed in DB2 10.1. For SAP systems we recommend that you set it to YES.

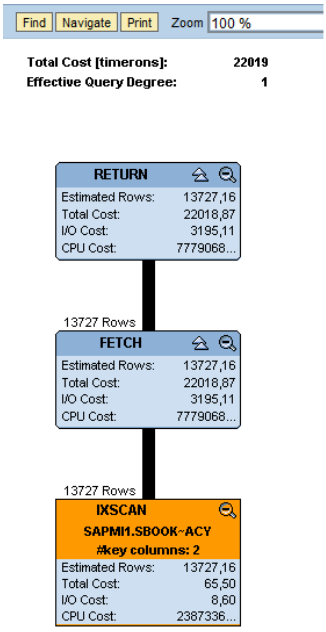
At the time of writing this article DB2 10.1 does not contain any monitor elements to count the occurrences of sequential prefetching and readahead prefetching.

Smart Prefetching in the EXPLAIN Output

The output of the *db2exfmt* utility was enhanced in DB2 10.1 to show which prefetch method the optimizer selected for a query. Possible values are

- NONE
- SEQUENTIAL
- SEQUENTIAL, READAHEAD
- READAHEAD

The following figure shows the graphical representation of the EXPLAIN output for a SELECT on SBOOK that uses the secondary index SBOOK~ACY which contains the MANDT and AGENCYNUM fields:



General Information	
Operator IXSCAN (#3)	
Catalog Information	
Predicates	
Costs	
Parameter	Value
Cummulative Total Cost	65,50
Cummulative CPU Cost	23873360,00
Cummulative I/O Cost	8,60
Cummulative Re-Total Cost	6,26
Cummulative Re-CPU Cost	23726118,00
Cummulative First Row Cost	6,92
Estimated Buffer Pool Buffers	9,60
Arguments	
Parameter	Value
PREFETCH	SEQUENTIAL READAHEAD
TABLOCK	INTENT SHARE
ROWLOCK	SHARE (CS/RS)
TBISOLVL	CURSOR STABILITY
MAXPAGES	8
SCANDIR	FORWARD
SKIPDROW	TRUE
SKIP_INS	TRUE
LCKAVOID	TRUE

General Information	
Operator FETCH (#2)	
Costs	
Parameter	Value
Cummulative Total Cost	22018,87
Cummulative CPU Cost	77790680,00
Cummulative I/O Cost	3195,11
Cummulative Re-Total Cost	9,55
Cummulative Re-CPU Cost	36218744,00
Cummulative First Row Cost	13,81
Estimated Buffer Pool Buffers	3242,37
Arguments	
Parameter	Value
PREFETCH	SEQUENTIAL READAHEAD
MAXPAGES	3186
TABLOCK	INTENT SHARE
ROWLOCK	SHARE (CS/RS)
TBISOLVL	CURSOR STABILITY
VISIBLE	FALSE
WRAPPING	FALSE
SPEED	SLOW
THROTTLE	FALSE
SKIPDROW	TRUE
SKIP_INS	TRUE
LCKAVOID	TRUE

The new argument “PREFETCH” shows for the index scan as well as for the fetch operation that the optimizer decides to start with sequential prefetching and possibly switch to readahead prefetching here.

There are also new output values available for the db2expln and dynexpln commands, see the IBM Information Center topic Table access information for details.

Summary

Every database table and its indexes have a physical representation on disk. DB2 stores table and index data in pages in a so-called table and index object. Following a high insert, update and delete activity the table object and the index object become fragmented. As data and index pages become more and more unclustered the prefetching algorithms used by DB2 9.7 or lower become less efficient. Furthermore, disk space trapped inside the table and index object is not usable for other tables and indexes within the same tablespace. The usual remedy to both problems is a table and index reorganization that is a costly operation and affects the availability of the table and its indexes to be reorganized.

DB2 10.1 introduces new mechanisms to reduce the need for reorganizations or make them completely unnecessary. To alleviate the problem of space reclamation, a lightweight index space reclaim operation is available for all indexes immediately after upgrading to DB2 10.1. Furthermore, the new table type insert clustering tables that is based on the existing MDC infrastructure allows to easily reclaim empty blocks in an online fashion without data movement if a specific access pattern (records that are inserted together are also deleted together) is used by the application. Existing non-ITC tables can be converted to ITC tables using the DB6CONV report. Moreover, DB6CONV can calculate candidates for ITC conversion based on the archiving statistics of an SAP system.

To improve the performance on unclustered tables and indexes DB2 10.1 offers a new prefetching technique called readahead prefetching which is in these situations more efficient compared to sequential prefetching. By using readahead prefetching exactly those pages that are needed by the query are fetched into the buffer pools. DB2’s optimizer is able to decide which prefetching technique is the most efficient one in a given situation. Selecting the right technique is called smart prefetching. It is enabled by default in DB2 10.1.

Independent of your DB2 version SAP Note 975352 provides meaningful guidelines on when a re-organization of tables and indexes is actually necessary.

Related Content

SAP Note 975352 DB6: Reorganizations in DB2/Using DB2 Auto REORG

SCN Article Reclaiming Space from Multidimensional Clustering (MDC) Tables

IBM Guide IBM DB2 Autonomics - A Practical Guide to DB2 Autonomic Features in SAP Environments

Copyright