SQL Server 2008 Technologies for SAP Solutions

Applies to:

NetWeaver 7.00 (SR3), 7.10 based products

Summary

This white paper article provides an overview about the new features of SQL Server 2008

Author: Jürgen Thomas, Principal Program Manager

Company: Microsoft Corporation

Created on: 14 May 2008

Author Bio

Jürgen Thomas is a Principal Program Manager at Microsoft Corporation.

Table of Contents

Introduction

Microsoft will release the new SQL Server release in summer 2008. SQL Server 2008 has some major new features, which will benefit SAP customers. The following paper introduces several of these new features. The features are divided into features where some adoptions have to be done on the SAP coding side and features, which are transparent to the SAP application stacks. Since there are features, which needed adoption and changes on the SAP side, precondition for SQL Server 2008 will not only be to run SAP applications based on NetWeaver 7.00, but there also will be a requirement for specific Basis Support packages. For more information, see SAP note 1152240.

New Features

SQL Server 2008 provides the following features that are actively leveraged by SAP coding:

- New ODBC client interface
- Data compression
- Star join optimization (Business Intelligence only)

The following features are helpful when running an SAP system. These features are transparent to SAP applications and hence need no further changes in SAP coding

- Backup compression
- Auditing
- Transparent data encryption
- Lock escalation granularity
- Logging enhancements
- Resource governor
- Query Plan freezing
- Enhancements around Database Mirroring
- Changed Autogrowth Algorithms

Features actively leveraged by SAP NetWeaver

ODBC Client Interface

With SQL Server 2008, SAP re-implemented the database interface and switched from OLEDB to ODBC.

The main reasons for this switch are:

- ODBC is the industry standard for database interfacing and therefore the mostly used and optimized technology
- The former OLEDB based interface has a ten years history starting with SQL Server 7.0. Since more features are offered by newer database releases and functionality moves from the interface into the database server, a lot of code becomes obsolete but has to be maintained for compatibility reasons.
- The new streamlined ODBC client improves the database performance from a client perspective by 3 to 5%.
- ODBC opens MS SQL Server access also for non-Windows SAP application servers. An official support has to be postponed after more intensive evaluation.

Data Compression

The main development goals of SQL Server 2008 Data Compression are to:

- Store numeric data more efficiently, like int, bigint, float and decimal data types
- Shrink table sizes of tables with larger parts of numeric data, but also optimize storage of default values
- Provide severe improved query performance in I/O bottleneck scenarios
- Reduce hardware requirements and operation cost around storage and backup infrastructure
- SQL Server 2008 Database Compression can be enabled on a per table/index granularity

With SQL Server 2008 there are two types of data compression available:

- ROW Compression - basically is nothing else than a new defined row format to optimize storing numeric values
- Page Dictionary Compression – working on a per page level storing common prefixes and values only once on the page

A preliminary first step in optimizing values stored in decimal data types was already introduced with SQL Server 2005 SP2. SAP supports this storage format introduced as VarDecimal for SAP BI. For details see: https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/a09bdc69-edd4-2910-b090-83ef3a1f4b51

ROW compression as delivered in SQL Server 2008 is an extension of what was delivered for the specific decimal case in SQL Server 2005 SP2.

The following figures 1 to 4 show an original page without data compression, data compression in SQL Server 2005 and SQL Server 2008.

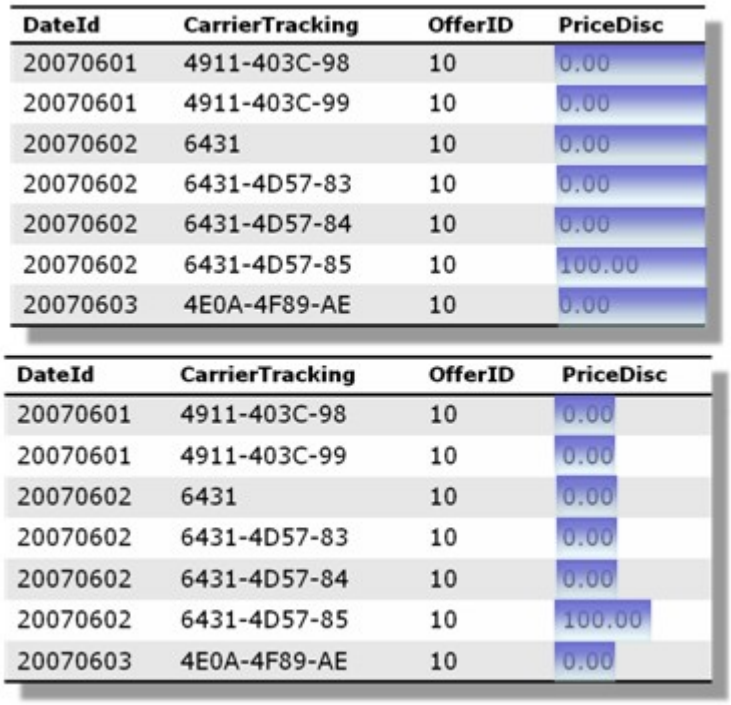
The first figure shows a page without compression.

DateId	CarrierTracking	OfferID	PriceDisc
20070601	4911-403C-98	10	0.00
20070601	4911-403C-99	10	0.00
20070602	6431	10	0.00
20070602	6431-4D57-83	10	0.00
20070602	6431-4D57-84	10	0.00
20070602	6431-4D57-85	10	100.00
20070603	4E0A-4F89-AE	10	0.00

Figure 1: Original Page without Data Compression

In all those cases a fixed length of bytes is used to store data of columns OfferID, which is an integer type column and the column PriceDisc which is a decimal type column. We know that the integer column takes 4 bytes on disk and the decimal column dependent on its definition can take up to 17 bytes on disk. Whereas the space consumed on disk is independent of the values assigned.

The second figure shows data compression that is already available as of SQL Server 2005. The VarDecimal storage compression allows decimal values to be stored as variable-length data.



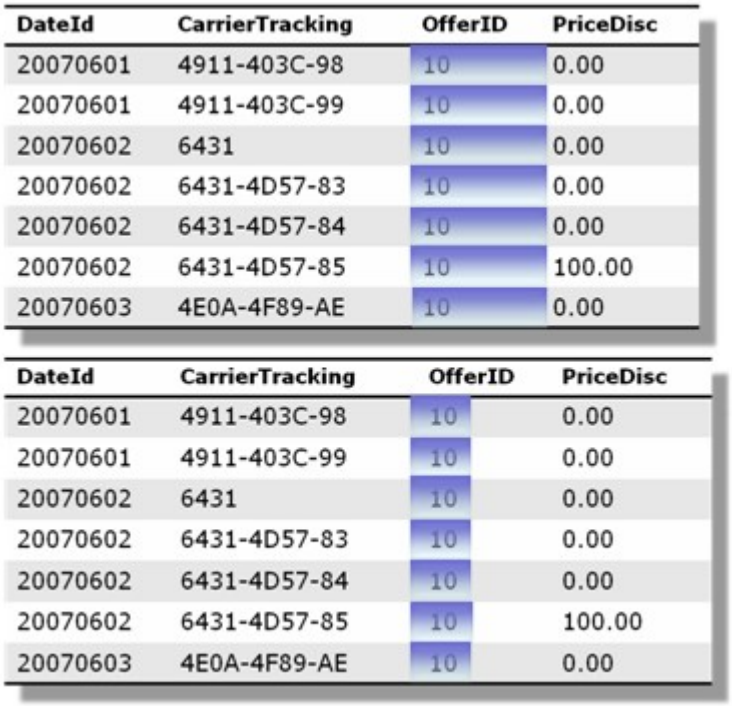
DateId	CarrierTracking	OfferID	PriceDisc
20070601	4911-403C-98	10	0.00
20070601	4911-403C-99	10	0.00
20070602	6431	10	0.00
20070602	6431-4D57-83	10	0.00
20070602	6431-4D57-84	10	0.00
20070602	6431-4D57-85	10	100.00
20070603	4E0A-4F89-AE	10	0.00

DateId	CarrierTracking	OfferID	PriceDisc
20070601	4911-403C-98	10	0.00
20070601	4911-403C-99	10	0.00
20070602	6431	10	0.00
20070602	6431-4D57-83	10	0.00
20070602	6431-4D57-84	10	0.00
20070602	6431-4D57-85	10	100.00
20070603	4E0A-4F89-AE	10	0.00

Figure 2: VarDecimal Storage Data Compression with SQL Server 2005 SP2

Instead of using a fixed length storage format, VarDecimal already allows the storage of decimal defined values dependent on the values submitted. For example, for a value of 0.0 only two bytes are used.

Figure 3 shows row compression with SQL Server 2008, which is extending the variable length storage format of decimal data type to other data types like int, bigint and float. Other optimizations also reduce the space needed to store values like 0 or 0.0. Instead like in VarDecimal consuming 2 bytes, such a value will only consumes 4 bits in SQL Server 2008.



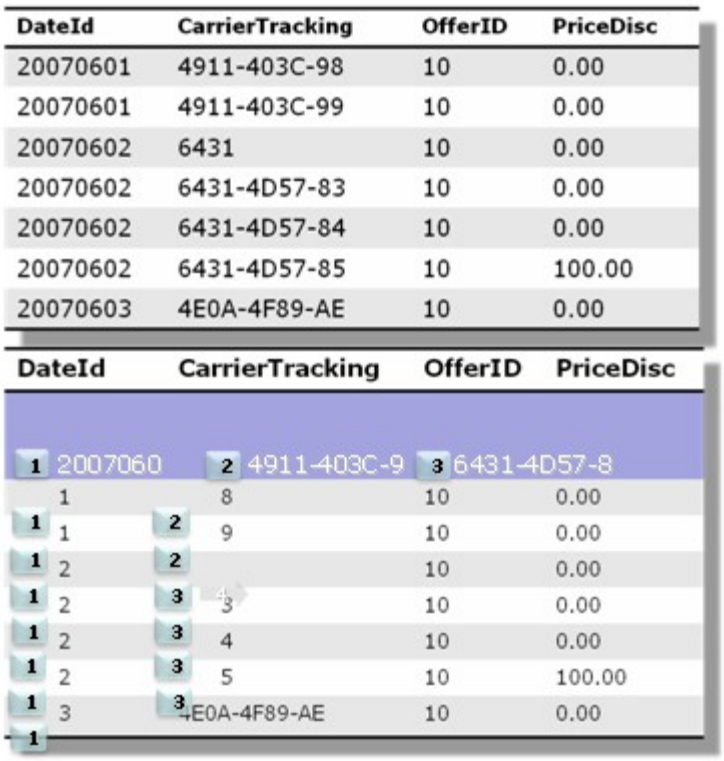
DateId	CarrierTracking	OfferID	PriceDisc
20070601	4911-403C-98	10	0.00
20070601	4911-403C-99	10	0.00
20070602	6431	10	0.00
20070602	6431-4D57-83	10	0.00
20070602	6431-4D57-84	10	0.00
20070602	6431-4D57-85	10	100.00
20070603	4E0A-4F89-AE	10	0.00

DateId	CarrierTracking	OfferID	PriceDisc
20070601	4911-403C-98	10	0.00
20070601	4911-403C-99	10	0.00
20070602	6431	10	0.00
20070602	6431-4D57-83	10	0.00
20070602	6431-4D57-84	10	0.00
20070602	6431-4D57-85	10	100.00
20070603	4E0A-4F89-AE	10	0.00

Figure 3: Row Compression with SQL Server 2008

The name ‘Row Compression’ is misleading in this case. In fact, a new row format is introduced, which is more sensitive on space consumption. But it has nothing to do with some compression algorithms trying to compress a row on a page.

Figure 4 shows page dictionary compression in SQL Server 2008, where a prefix list is stored in the page for common prefixes. Individual values are replaced by a token for the prefix, and a suffix for the value.



DateId	CarrierTracking	OfferID	PriceDisc
20070601	4911-403C-98	10	0.00
20070601	4911-403C-99	10	0.00
20070602	6431	10	0.00
20070602	6431-4D57-83	10	0.00
20070602	6431-4D57-84	10	0.00
20070602	6431-4D57-85	10	100.00
20070603	4E0A-4F89-AE	10	0.00

DateId	CarrierTracking	OfferID	PriceDisc
1 2007060	2 4911-403C-9	3 6431-4D57-8	
1	8	10	0.00
1 1	2 9	10	0.00
1 2	2	10	0.00
1 2	3 3	10	0.00
1 2	3 4	10	0.00
1 2	3 5	10	100.00
1 3	3 4E0A-4F89-AE	10	0.00
1			

Figure 4: Page Dictionary Compression with Individual Values in SQL Server 2008

Page dictionary compression has two optimizations, which apply to a data/index page of SQL Server. First step is to find prefixes which are common in different rows. A classical example, which fits very well for the SAP case, is the DateID, which pretty much represents the way SAP applications store dates in a varchar column. As it is obvious in this example, there is a common prefix of ‘2007060’ to all the values in the column DateID. SQL Server page compression stores this prefix value separately on the page and places a 1-2 byte token as a placeholder pointing to the shared prefix.

Figure 5 shows page dictionary compression in SQL Server 2008, where a common value dictionary is stored in the page. Common values are replaced by tokens.

Figure 5: Page Dictionary Compression with Common Values in SQL Server 2008

In the second step common values are replaced by a token. This is only done if some space savings can be achieved.

Experiences made with Data Compression in SQL Server 2008

The following provide a short overview of the experiences made so far with row and page dictionary compression.

ROW Compression:

Note: Row compression will become the default compression type for SAP installations.

The following experiences were made with row compression so far:

- Reduction of space of completely re-organized SAP ERP database without any fragmentation by 15%
- Reduction of space of a real live 5.5TB SAP ERP Unicode database by 32% (includes effects of data re-organization)
- More space savings are expected for specific IS solutions like IS-U or banking solutions due to large tables with numeric data
- Reduction of size of completely re-organized SAP BI without any fragmentation by around 35%
- No measurable negative performance impact. First indications even point to lower CPU consumption.
- Usually positive performance impact due to reduction of I/O especially in SAP BI

- Performance improvements are more obvious the more the system is bottlenecked on I/O subsystem
- Hardly any higher CPU consumption measurable

Page-Dictionary Compression:

Note: SAP will offer Page-Dictionary Compression for specific tables.

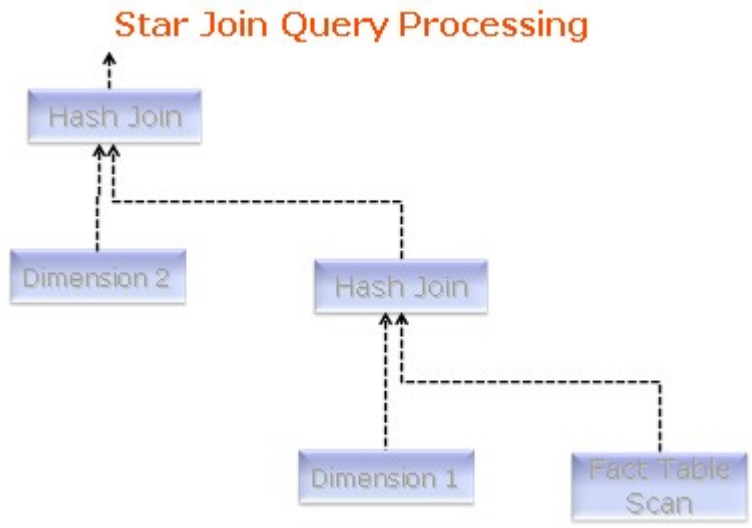
The following experiences were made with Page-Dictionary Compression so far:

- When applied on all tables of completely re-organized ERP the size of SAP ERP can be reduced to 50% of its original size.
- Tremendous effect on tables like GLPCA with shrinking table down to 15% of original size
- Other tables are hardly shrinking (e.g. customizing tables)
- Not advisable to use for Queuing tables like VBHDR, VBMOD, VBDATA, ARFCSSTATE, ARFCSDATA, TRFC…
- Measureable negative performance impact on CPU in ERP by around 20-25% higher consumption in case all ERP tables were compressed (except Queuing tables).
- In general, improved performance in extreme I/O bottlenecked scenarios

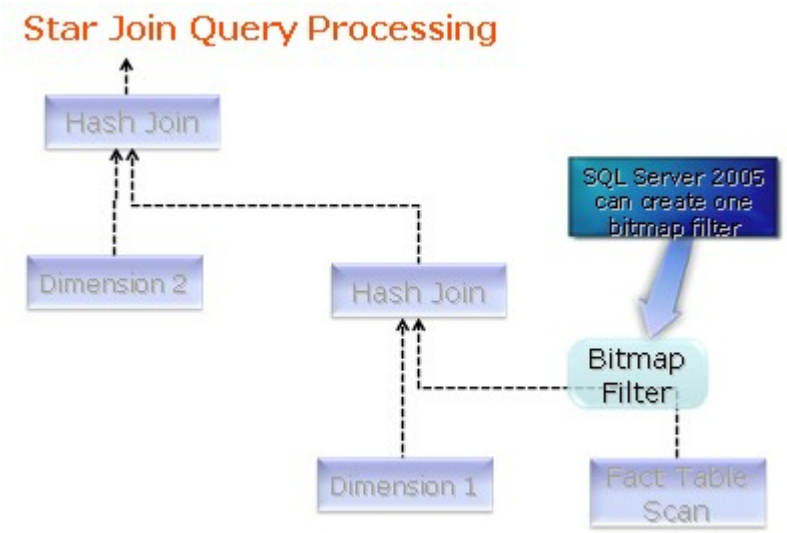
Star Join Query Processing

SQL Server 2008 introduces a new way to build query plans around Star or Snowflake schemas. Schemas as experienced in SAP BI. One of the typical observed behaviors so far was that SQL Server would join dimension by dimension with the fact table or the intermediate result set of the former join result. But what hardly was observed was SQL Server analyzing several dimension tables first, overlaying the matching rows and then accessing the fact table to perform a first join.

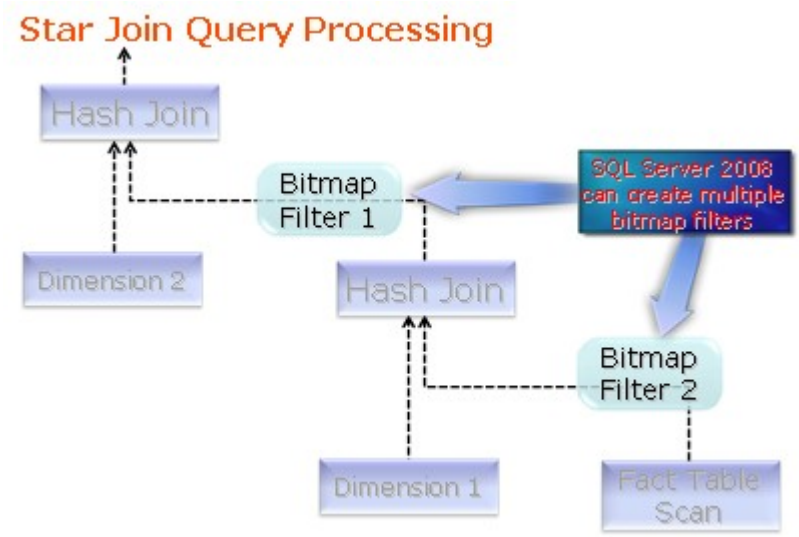
This is demonstrated in the next sequence. A typical plan as observed today looks like joining the dimension, which promises the best reduction of rows with the fact table and then take the intermediate result set and join that with the next dimension.



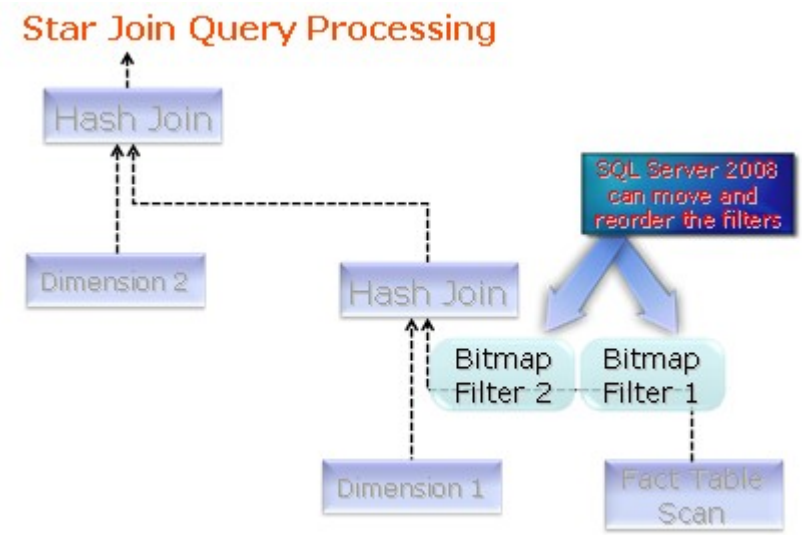
For this purpose SQL Server 7.0, SQL Server 2000 and 2005 built one dynamic bitmap filter, which was usually used for the first join.



With SQL Server 2008 multiple of these dynamic bitmap filters can be built.



What is more important is that these dynamic bitmap filters can be moved and so to speak overlaid before accessing the fact table. This means a combined filter of multiple dimension tables is applied in one step.



The backdrop is that this new optimization for Starjoin Processing only works with a max degree of parallelism, which is >1. However our experience about enabling parallel execution of queries was that system behavior is less predictable. Therefore the parallelization is switched on for very specific BW Star Joins only. These changes are subject of a SAP BW Support Package (we need to get the OSS note number in here). The plan is to have some BW queries executed with a degree of parallelism of 2 and to test this new feature with some of our TAP and Beta Customers.

Transparent Features of SQL Server 2008

Backup Compression

Backup compression is one of the features that are in high demand by many customers. So far there have been only third party solutions providing a chance to have a SQL Server backup compressed. Motivation for most of the customers asking for such a feature is the fact that backups should be kept on disk and/or that such backups should be transferred to centralized backup infrastructures. With SQL Server 2008, the capability of compressing any type of SQL Server backup comes out of the box.

SQL Server 2008 offers the following advantages:

- Has WITH COMPRESSION clause to BACKUP
  `backup database E61 to DISK = N'D:\E61full_compress.bak' with COMPRESSION`
- Less storage required to keep backups online
- Backups run significantly faster, as less IO is done
- Restore automatically detects compression and adjusts accordingly
- Integrated into Resource Governor to throttle CPU consumption
- Backup Compression can be used by SQL Server Logshipping functionality
- Tests so far reveal parity with SQL Litespeed
- Usable by all 3rd party backup vendors via VDI or VSS interface

Contrary to SQL Litespeed, SQL Server 2008 backup compression does not have any compressions grades to configure. It is one grade, which is possible with SQL Server. When comparing the results between different grades of SQL Litespeed and SQL Server 2008 backup compression with a 5.5TB SAP Unicode ERP, the following statements can be made:

- SQL Server 2008 backup compression provides a better compression rate with slightly more CPU consumption than SQL Litespeed that is configured for compression grade 1
- SQL Litespeed provides a slightly better compression rate with compression grade 11 and significantly more CPU resource consumption than SQL Server 2008 Backup Compression
- Transaction log compression rate is between 2.0-2.5
- Compression rate on SAP ERP is between 4.0-5.0

Audit

Another new functionality in SQL Server 2008 should address auditing of user activity. Whereas from the beginning this functionality was not meant to allow auditing database accesses of different users logged into one of the SAP applications. This functionality should address the compliance and auditing concerns on who else than your SAP users are accessing the database directly not using any one of the SAP applications. At the end it boils down to be able to monitor the activities of operators or even DBAs on SAP databases. For this kind of purpose the new audit functionality was created as what one calls a first class server object. This means one doesn't audit using SQL Server Profiler, but one can define audit sessions and audit rules by simple T-SQL statements. No additional tools are necessary. The functionality is completely integrated into SQL Server's Relational Engine

Figure 6: Move Capabilities of Different Tools into one SQL Server Engine Functionality to Audit Activities

However, to make auditing of non-SAP motivated database accesses successful in an SAP landscape, you have to keep a few things in mind:

- Never keep the log on the database server
- SAPServiceSID, SIDAdm are not allowed to have access to the share on the server where log resides
- None of the DBAs may have access to the share on which the log resides
- The BUILTIN\Administrator Login in SQL Server needs to be deleted
- SQL Server should run in the context of an account, which is not allowed to login interactively

Of course, you cannot avoid that somebody identified in the sysadmin role, can disable the audit session or change its definition. However, you can build an audit session, which tracks:

- Who logged into SQL Server and got mapped into the sysadmin role
- What changes were done to the audit session with that logged in session

Given the fact that the very same person does not have access to the audit log, it should not be possible for that person to manipulate the results of the audit log showing manipulations.

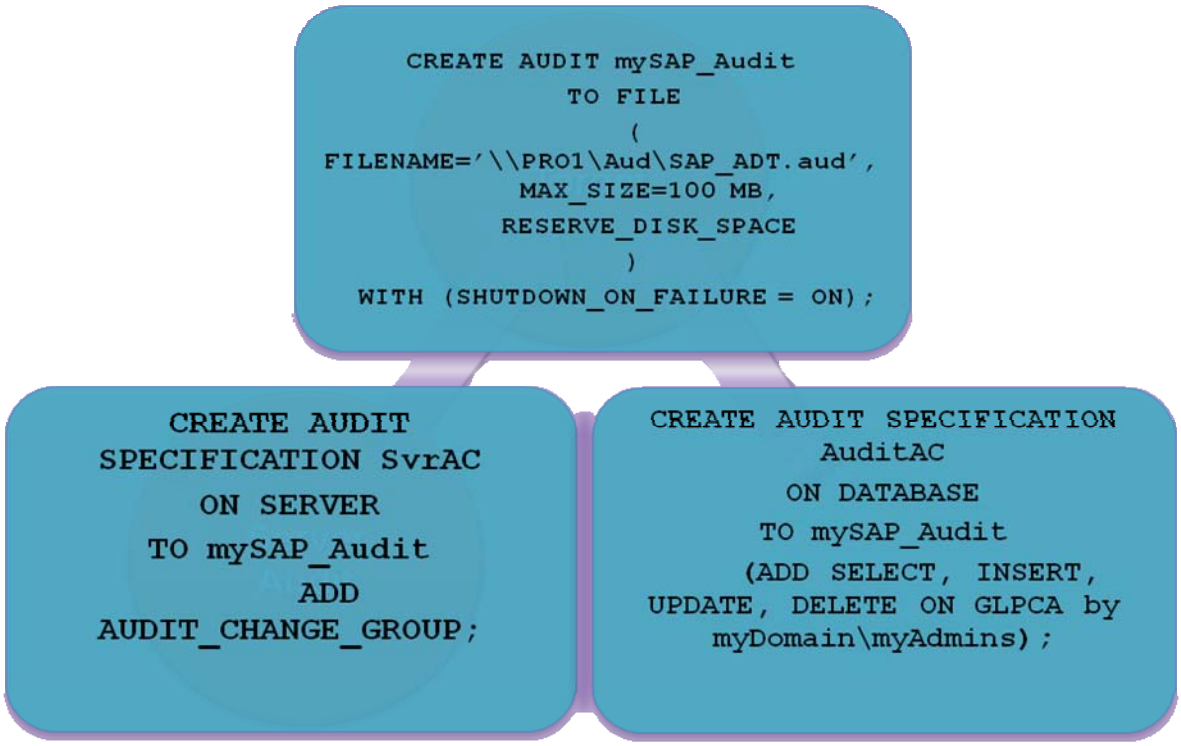


Figure 7: A Very Simple Audit Session and its Specifications Defined

Transparent Data Encryption

SQL Server introduced encryption of columns within a database already with SQL Server 2005. However to applications this type of column encryption was not transparent at all. In cases of some applications significant changes would have been required to leverage column encryption besides some other disadvantages of not being able to leverage index searches on such columns. In SQL Server 2008 a new functionality has been introduced that is called Transparent Data Encryption. In this case the scope of encryption is the whole database. The encryption itself will be transparent to an application since accessing or not accessing an encrypted database is an affair of the specific SQL Server instance. Or in other words, once a SQL Server instance can open an encrypted database, every application or user can access the data in this database. This also makes clear that this kind of transparent encryption is not meant as measure of user access control or to keep some users off from reading content in the database. This is a very important difference to the column level encryption introduced with SQL Server 2005 which allows these type of access controls. The encryption/decryption of Transparent Data Encryption is done in the I/O path. This means that the data is in an non-encrypted state in the SQL Server Buffer Pool and hence can fully be leveraged for indexing and index searches.

The following summarizes the capabilities and experiences of Transparent Data Encryption so far:

- 4 different Encryption Algorithms
  AES_128, AES_196, AES_256, TRIPLE_DES
- Enabling Encryption encrypts all new pages written to disk PLUS will trigger background threads to encrypt rest of database
- Best throughput and least overhead with AES algorithms
- Not recommended to use TRIPLE_DES since it is the weakest Encryption algorithm and is not as efficient in resource consumption as the AES algorithms
- Overall impact hardly measurable
- Backing up an encrypted database will automatically provide an encrypted backup
- Hardly any backup compression achievable on an encrypted database using SQL Server Backup compression or SQL Litespeed
- SAP specific whitepaper will be released before SQL Server 2008 RTM

Lock Escalation Granularity

The locking strategy of SQL Server is called dynamic locking. This means that in some cases, SQL Server in its relational engine tries to be smart in terms of using a specific lock granularity. What SQL Server wants to avoid is that the major space in the buffer pool is taken over by Lock structures. Therefore, SQL Server versions released so far had a few events where the lock granularity could be escalated to a higher granularity. The events roughly look like:

- 40% buffer Pool used by lock structures. In the SAP space this usually means a few million locks
- More than 5000 row locks held by DML statement on one table. In this case there is an attempt to escalate to Table lock granularity. If there is another transaction with other locks on the table, one will continue with acquiring more row level locks. This type of escalation attempt can be suppressed by trace flag 1211 or 1224
- Storage Engine decides to start a statement with Page-Level locks. You can disable this with sp_indexoption
- Storage Engine decides to start a statement with Table-Level locks. You cannot disable it.

Obviously one of the issues is that there is not a central switch where one could tell not to escalate at all and always to use row level locks for a specific table. This functionality was added to SQL Server 2008 with a command like this:

```
ALTER TABLE VBDATA set (LOCK_ESCALATION = DISABLE)
```

Minimal Logging Enhancements

Several changes around the way logging is done were introduced in SQL Server 2008. The most significant changes have impact in Unicode or platform migrations of SAP applications. One of the current issues observed when performing Unicode migrations with database of volumes in the Terabytes was the consumption of a lot of transaction log space. The series of events leading to this higher consumption of transaction log space read like:

- Loading tables in SAP installation or Unicode/platform migration order is like:
  - Creating empty table
  - Creating Clustered Index
  - Loading data in batches of 10,000 rows
  - Creating non-clustered indexes
- Effect of the above sequence:
  - Only first batch is minimal logged
  - All subsequent batches get fully logged. This means, millions of rows are entered into the transaction log as well
  - Especially in Unicode/platform migrations a huge volume in SQL Server Transaction Log is required

SQL Server 2008 expands the minimal logging behavior to all batches loaded. So far this was observed to cut down the required space for the SQL Server transaction log by a huge factor of up to 50.

Query Plan Freezing

In SQL Server 2005, Microsoft introduced a feature called Plan Guides. This feature should have helped to get around one of the paradigms current databases have. On the one side a DBMS needs to cache access plans (query plans) since generating these on the fly simply cannot be done with 5000 or 10000 statements being executed per second. On the other side skewed data or extreme uneven data distribution in some columns of tables do not guarantee that an access plan created for one set of values submitted to a query, fits perfectly for a different set of values. The idea was to have a possibility to influence the generation of an access plan from the outside without having to change the application by introducing query hints. However experience made with plan guides showed that the feature was very hard to work with and hence hardly used in the SAP space so far.

Therefore, the generation of such plan guides were expended and made easier. SQL Server 2008 provides a way to create plan guides or better to freeze existing query plans based on plan_handles, which can be found a DMV called sys.dm_exec_query_stats. For more information and queries, see: http://blogs.msdn.com/saponsqlserver/archive/2007/06/05/sap-dbacockpit-and-some-related-sql-scripts-part-1.aspx

The typical use case in the SAP space for this feature is:

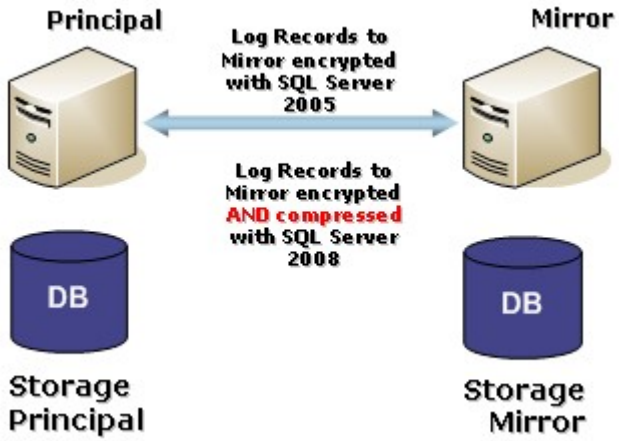
- Not to allow plan changes for critical plans
- Troubleshooting queries (cardinality tuner)
- Plan fixing for queries accessing tables/columns with extreme skewed data

Query Plan Freezing works as follows:

- Check sys.dm_exec_query_stats to find the plan to be frozen
- Find queries to do so on: http://blogs.msdn.com/saponsqlserver/archive/2007/06/05/sap-dbacockpit-and-some-related-sql-scripts-part-1.aspx
- Take a plan_handle and check whether the query has the plan to be frozen
- Take sql_handle and statement offset and execute
  ```
  sp_create_plan_guide_from_cache @name = N'myfirst_plan_guide', @plan_handle =
  <sql_handle>, @statement_start_offset = <statement offset>
  ```

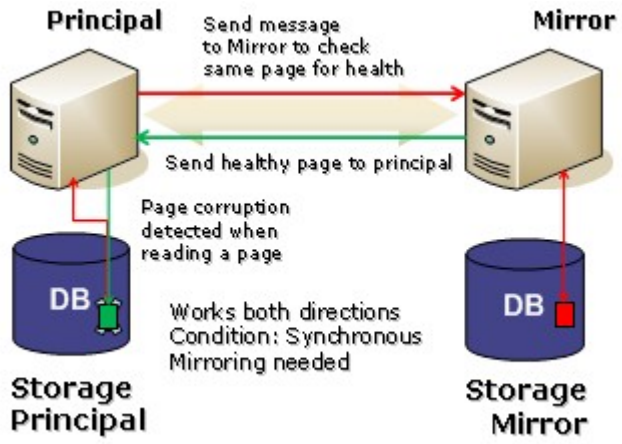
SQL Server Database Mirroring Enhancements – Compression

SQL Server Database Mirroring is more and more used in the SAP space. Larger customers meanwhile use this method very successfully. For SQL Server 2008 only slight but impactful changes were made for SQL Server Database Mirroring. The first change is that after encrypting the transferred data, the data becomes compressed. The experience is that this compression reduces the requirement on network bandwidth quite a bit.



SQL Server Database Mirroring Enhancements – AutoRepair

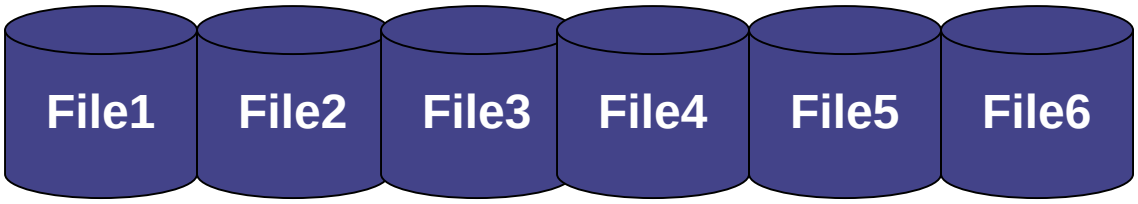
Another feature added to SQL Server 2008 is called AutoRepair of pages. This functionality catches cases where SQL Server detects a consistency issue on one or multiple pages. In the case of synchronous mirroring, Database Mirroring now tries to check the other system whether the very same page on that system is healthy. If this is true, Database Mirroring is exchanging the inconsistent page of the one system with the healthy one from the other side. This way of AutoRepair does work from the principal detecting an inconsistent page in its database and then getting the healthy page from the mirror. But it also works the other way round. As condition the Database Mirroring configuration needs to be synchronous. AutoRepair does not work with asynchronous database mirroring.



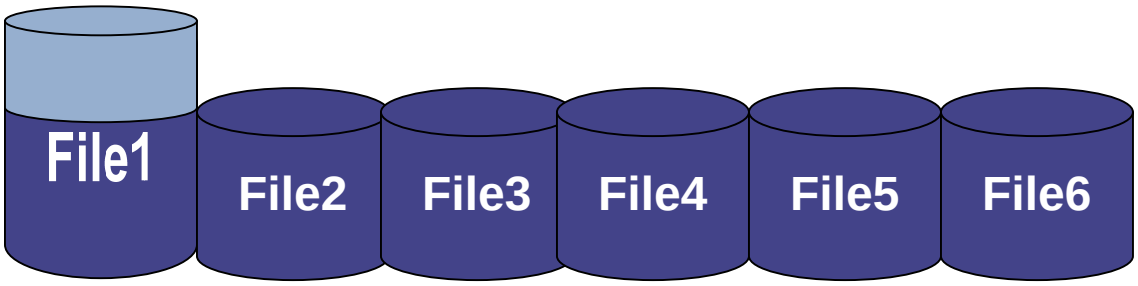
Changed Autogrowth Behavior

One of the features of various SQL Server versions in the last 10 years which did not work very well with the way how SAP applications are ported to SQL Server was the autogrowth behavior of SQL Server. The basic issue becomes obvious with the following few demonstrations:

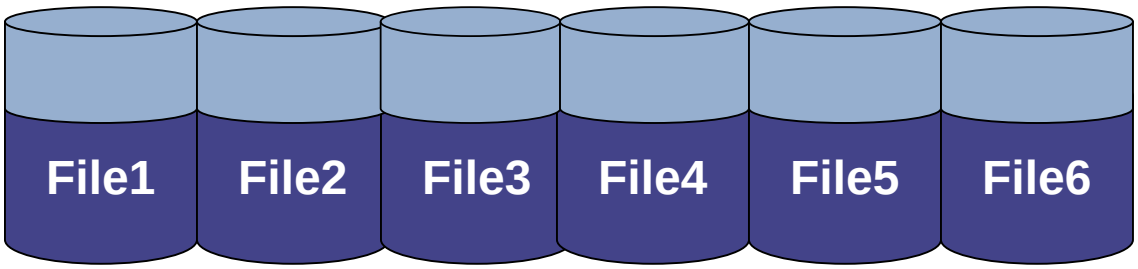
Assume the database looks like:



All the data files have the same size as desired and the files are getting filled in even portions as desired. However, if the files are running full, the current algorithms will grow one file only like this:



This kind of behavior destroys the even fill that we would like to see over the different data files. SQL Server 2008 introduces startup trace flag 1117. This trace flag changes the autogrowth behavior to one where all data files are grown in the same size (assuming the autogrowth rate is set the same for each of the files) instead of growing one data file only.



Related Content

http://www.sdn.sap.com/irj/sdn/mss

http://www.microsoft.com/isv/sap/technology/platform/sql.aspx

http://blogs.msdn.com/saponsqlserver/

Disclaimer and Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.