SAP® MaxDB™

Expert Session – Logging

MaxDB/liveCache Development Support
June 2010

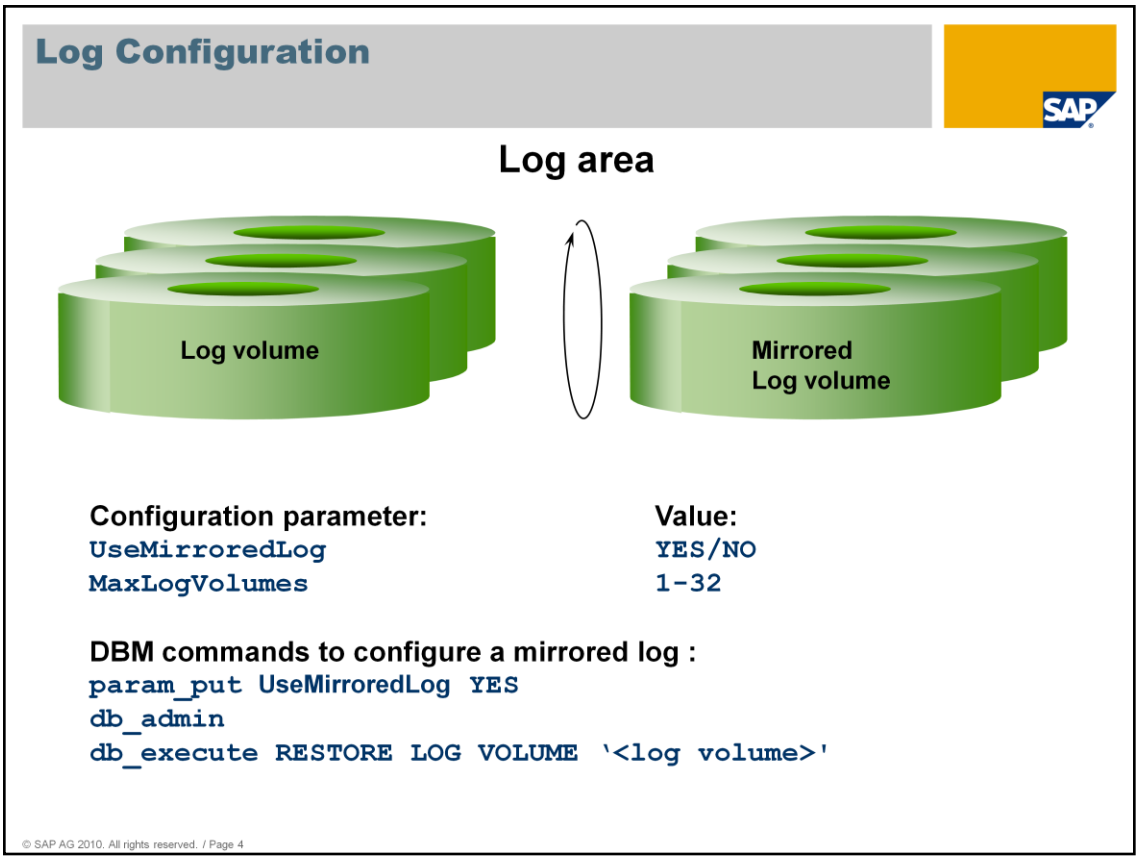
THE BEST-RUN BUSINESSES RUN SAP™

Expert Session
Logging

MaxDB/liveCache Development Support
Heike Gursch
Oksana Alekseious
June 29, 2010

Agenda

1. **Log Configuration**
   - Log Backups
   - Log Volumes
   - Overwrite Mode/Disabling Logwriter
   - Log Partitions
2. **Some Theoretical Aspects**
   - Redo and Undo Log Entries
   - Log Queue Management
   - DDL and DML Statements
3. **Transaction Handling**
   - Savepoints
   - Automatic Recovery during Restart
4. **Homogeneous System Copy/Shadow Instance**



In the basic configuration, SAP MaxDB writes all data changes to the log area. The log area consists of 1 to 32 log volumes.

The log area is overwritten in cycles. Before it can be overwritten, an area must be backed up.

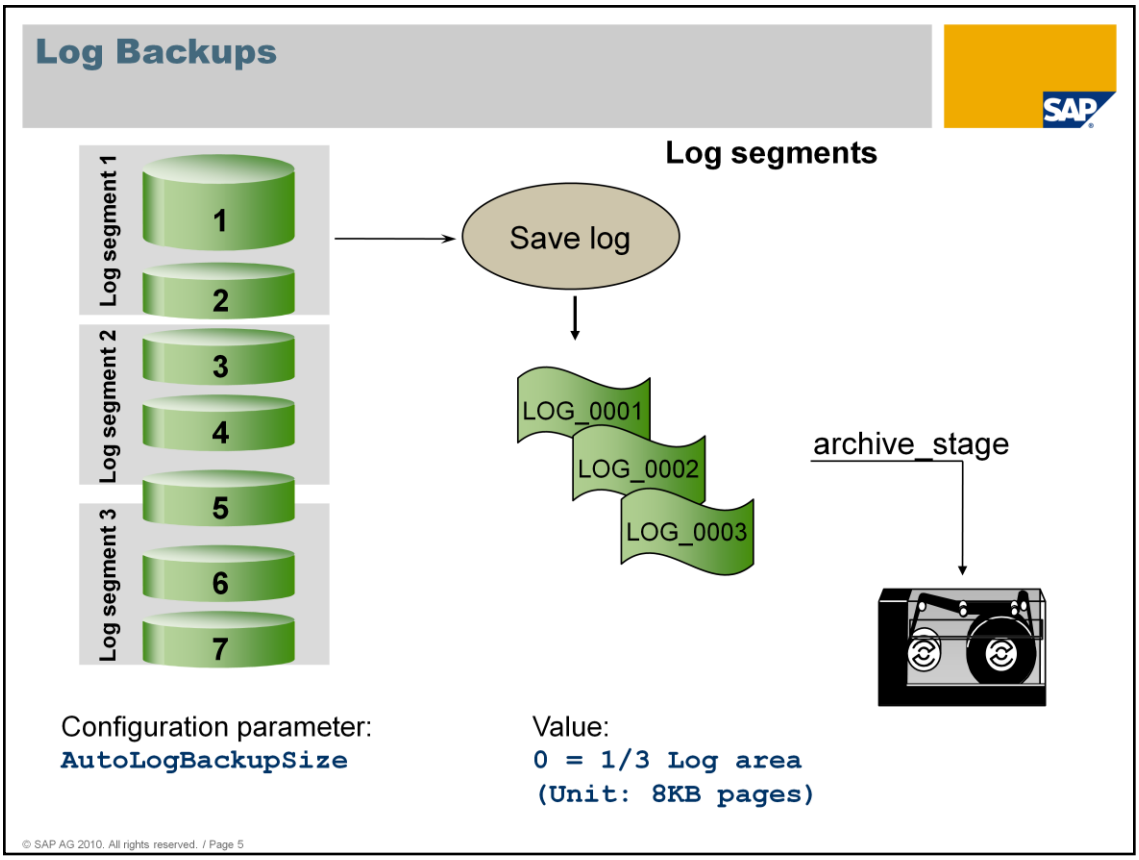
In systems in which the database has to ensure that data is persistent, the log volumes must be mirrored. This mirroring can be done by MaxDB. If the parameter UseMirroredLog is set to NO, the log must be completely mirrored with operating system functions or by the hardware (RAID 1).

- **If there is no copy of the log, disk errors result in loss of data.**

For security and performance reasons, RAID 5 is not recommended.

The parameter MaxLogVolumes indicates the maximum number of log volumes. The log areas can be expanded up to this number in online mode.

If the log area is mirrored by the database, the failure of a log volume causes an emergency shutdown. A new disk is then needed. The new log volume can be integrated in the ADMIN state.



The log is divided into fixed-length segments.

This segmentation does not represent a physical division of the log.

Log segments are backup units. They specify the size AutoLogBackupSize of the files in which the log pages are backed up. The database parameter AutoLogBackupSize indicates the size of a log segment in log pages.

If the size 0 is entered for the kernel parameter, the database manager automatically calculates 1/3 of the of the total log area. If the log is expanded online, this does not change the size of the log segments.

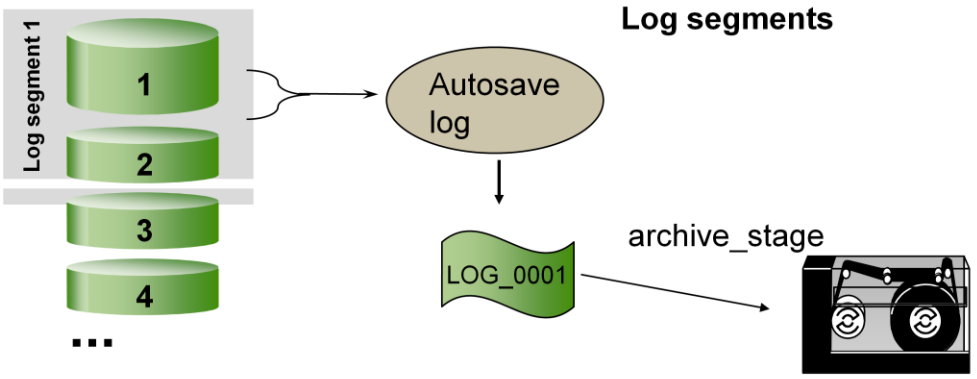
Log segments are NOT closed by a savepoint.

Interactive log backup backs up all completed segments (segment-by-segment) and the log pages of the segment that is not set complete. So there can be backup files that are smaller than AutoLogBackupSize.

The segment size cannot be more than half of the log area.

SAP recommends saving log backups in files in a file system. These backup files can then be transferred to a backup tape with the Database Manager command archive_stage. External backup tools are supported.

Automatic Log Backup



Configuration parameter: `AutoLogBackupSize`

Value: `0 = 1/3 Log area`

DBM command:

```
autolog_on [<medium>] [INTERVAL <interval in seconds>]
```

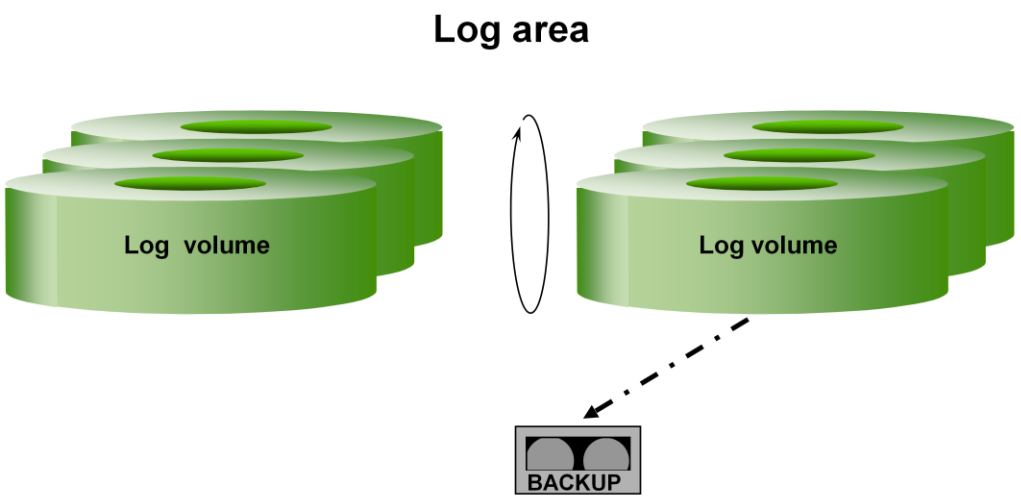

The database kernel can create the log backup automatically.

You activate automatic log backup with the dbmcli command autolog_on.

The log backup is automatically created asynchronously upon completion of a segment.

Also a time interval may be set to launch the automatic log backup along this interval.

Overwrite Mode for the Log Area (without Backup)



DBM command (in state ONLINE or ADMIN):

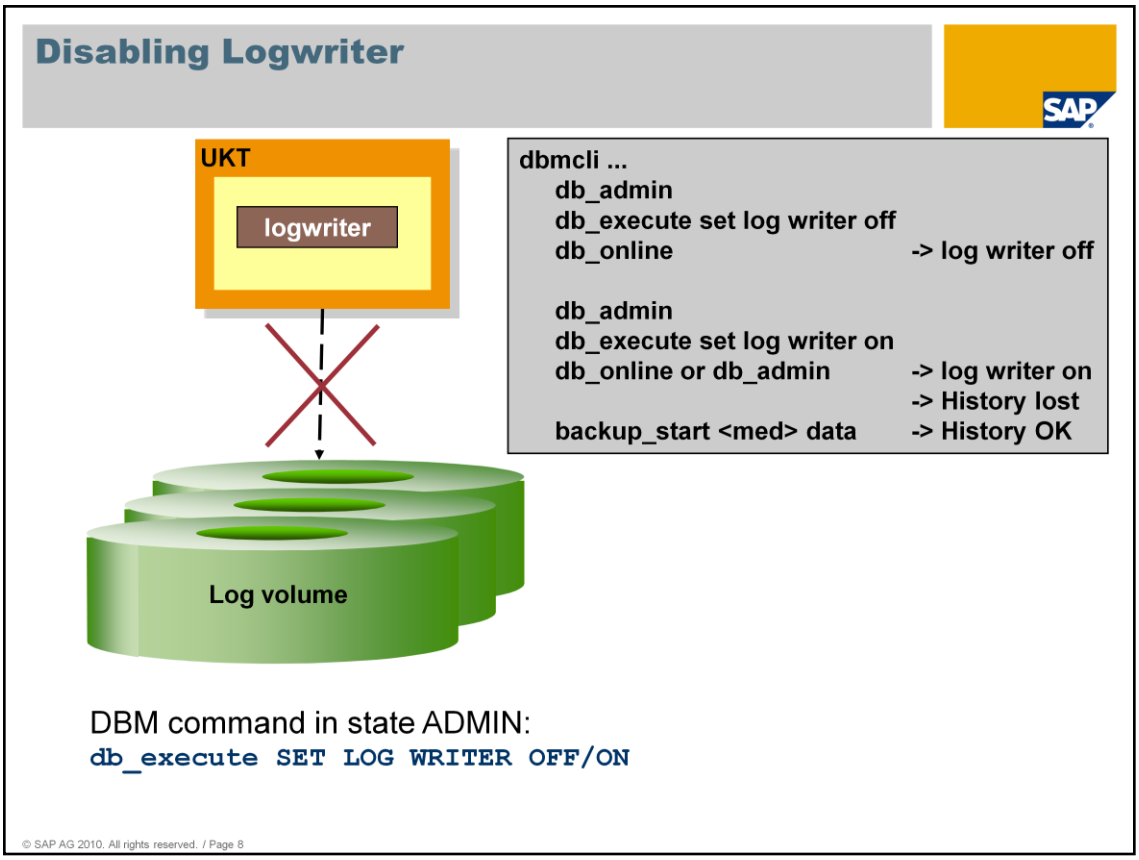
```
db_execute SET LOG AUTO OVERWRITE ON/OFF
```


The log can be overwritten automatically without log backups. Use the DBM command „db_execute SET LOG AUTO OVERWRITE ON“ to set this status.

Log backups are not possible after activating automatic overwrite. The backup history is then interrupted; this is indicated in the backup history by the identifier HISTLOST (file dbm.knl).

The backup history starts again when you deactivate automatic overwrite without log backup using the command „db_execute SET LOG AUTO OVERWRITE OFF“ and create a complete log backup in the ADMIN or ONLINE state. Remember to reactivate automatic log backup when this is desired.

Automatic overwrite of the log area without log backups is NOT suitable for production operation. As there is no backup history for the subsequent changes in the database, it may not be possible to redo transactions in case of a recovery.



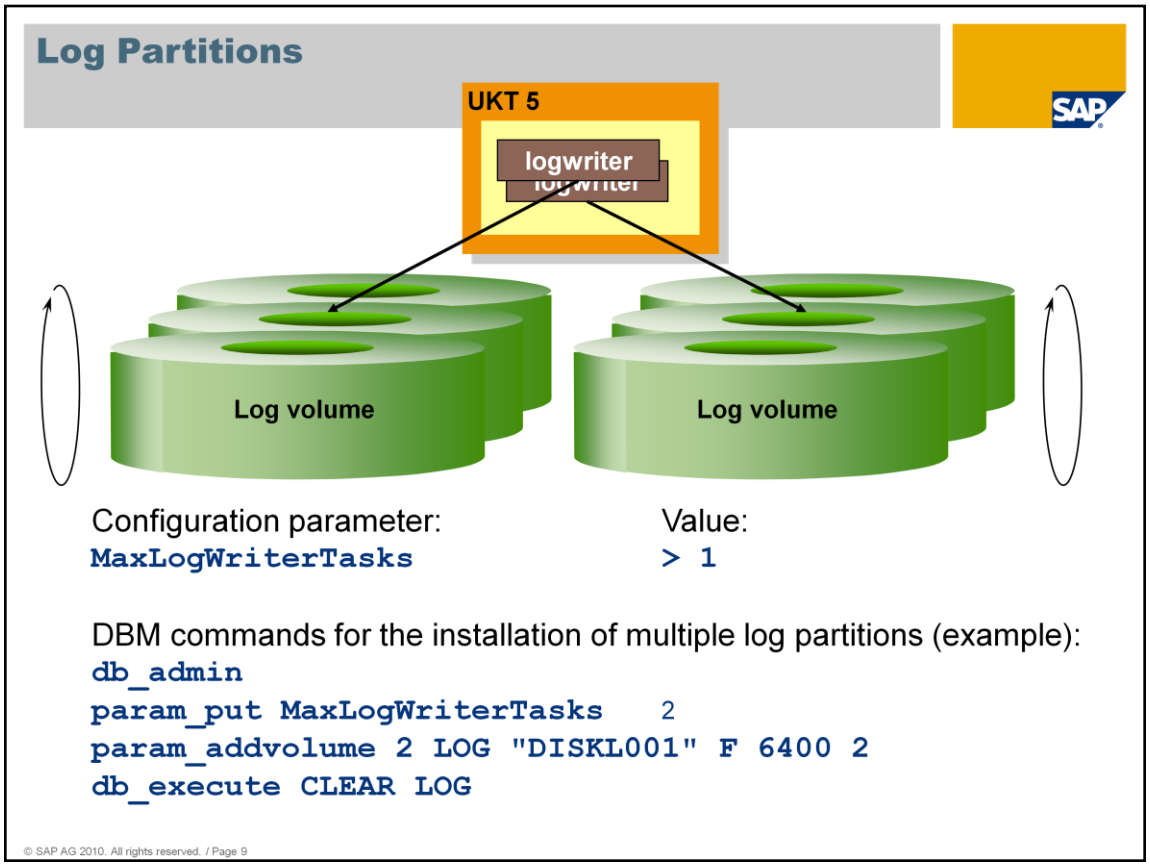
As the database is made consistent with each savepoint, it is possible to switch off the log writer. A restart is possible without the log because all open transactions can be redone on the basis of the last savepoint.

You switch the log writer off with the DBM command „db_execute SET LOG WRITER OFF“. To do so, start the database in the ADMIN state.

Log backups are not possible after the log writer has been switched off. The backup history is interrupted, as indicated by the identifier HISTLOST.

The backup history restarts when you reactivate the log writer with the DBM command “db_execute SET LOG WRITER ON“ in the state ADMIN and create a complete log backup in the ADMIN or ONLINE state. Remember to reactivate automatic log backup when this is desired.

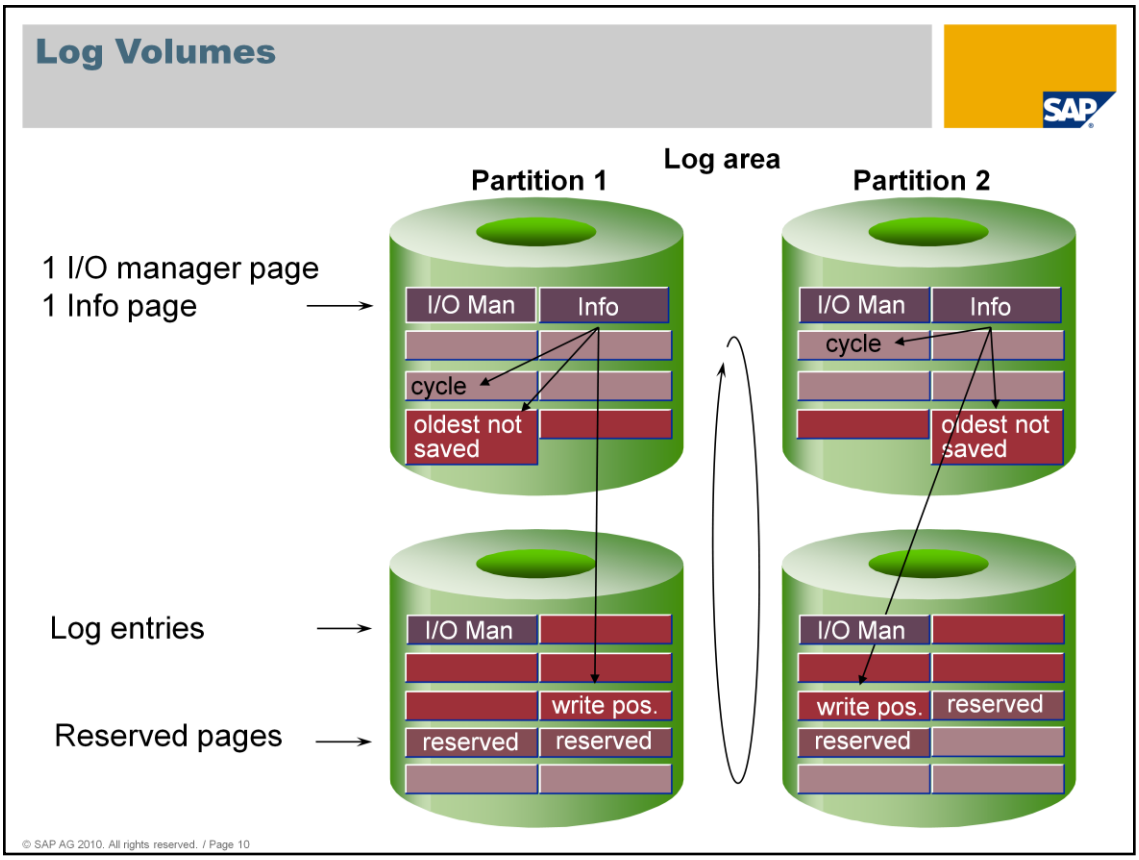
The log writer may not be switched off for production operation. The function serves to accelerate administration tasks such as upgrades and big load jobs.



As of version 7.8 MaxDB allows the use of multiple log partitiions. With parallel writing to the log volumes the database prevents bottlenecks during the access to the log queue and additionally wait situations for writes into the log volumes.

Partitions and also volumes of the partitions may have different sizes.

Normally user tasks of a UKT are assigned to a special log writer and therewith to a partition. This implies that for some tasks the state „log full“ might occur even if there is still some free space for a user task of another UKT in the corresponding log partition. Perform a backup of the log area if the state „log full“ is shown.



Log pages are always 8KB. They are written from the memory to the volume when they are full or when a log entry ends a transaction (COMMIT, ROLLBACK).

The log area is overwritten in cycles. Log pages can only be overwritten when they have been backed up. (Exception: SET LOG AUTO OVERWRITE ON).

In the header of the log there are two pages that are not cyclically overwritten.

The first page contains information from the I/O manager, for example the volume number and the numbers of the predecessor and successor volumes, if available.

Positions are administered on the info page. These include:

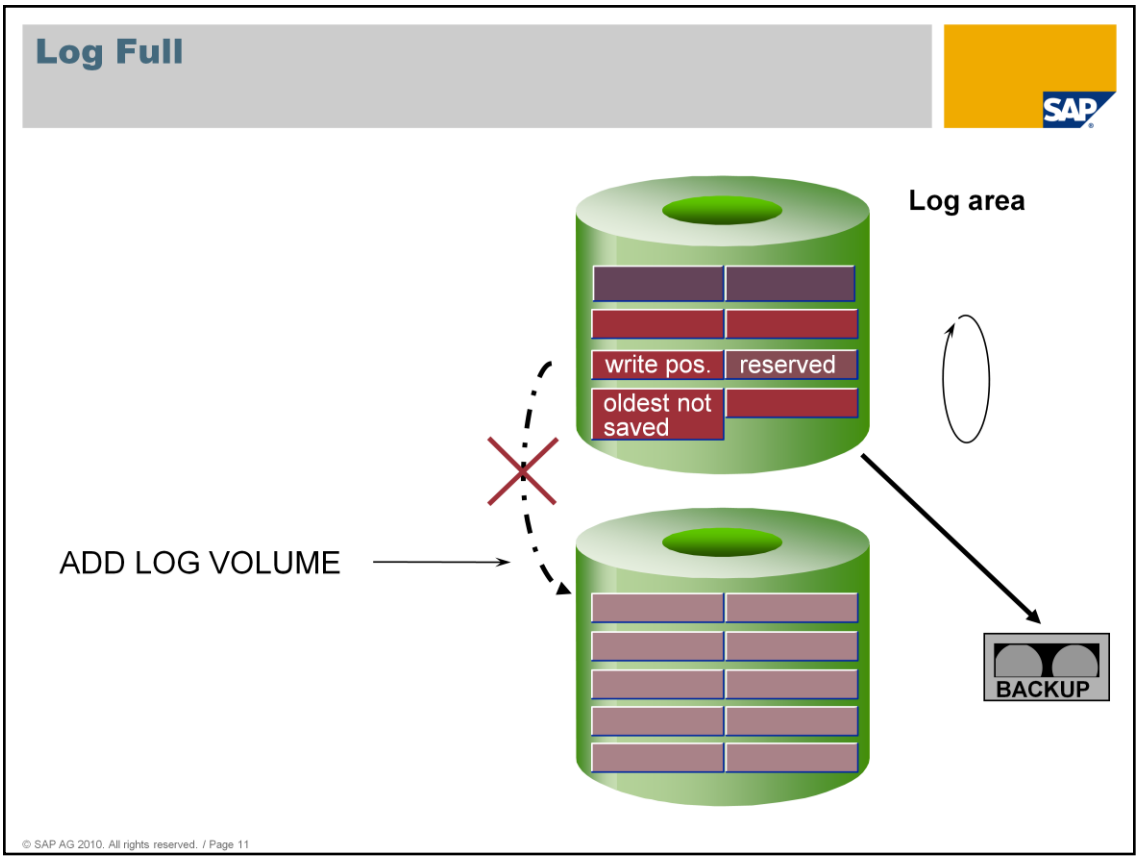
- the current write position,
- the position to which overwriting can proceed,
- the position that has not yet been backed up,
- the number of the last log backup.

The info page is written to the volume with each savepoint and, as a precaution, every 500 log pages.

If the current write position reaches the page up to which may be overwritten, the log is full. In this case, the database allows no further write transactions. Clients can still log on, as a connect is not written in the log.

The reserve pages cannot be filled with normal log entries. Following a log full status, at the next successful restart the first unused page is filled with a savepoint.

Each partition keeps its management information separately.



When the log has reached its capacity, creating a log backup is the only way to continue working in online mode. A log backup can be created in the ADMIN or ONLINE state. A log backup can also be created by activating automatic log backup.

If the database has the Log Full status, this is not reset by an expansion of the log area. The current write position is merely moved sequentially. It is not moved from the middle of a log volume to the beginning of the subsequent log volume.

If the log area is expanded through the addition of a new log volume, the new volume is only written when the current write position comes out of the last page of the predecessor volume.

When the Log Full status occurs, the database kernel writes a savepoint. Log entries can only be overwritten if they have been backed up and the position of the last savepoint in still in the log area (Exception: SET LOG AUTO OVERWRITE ON).

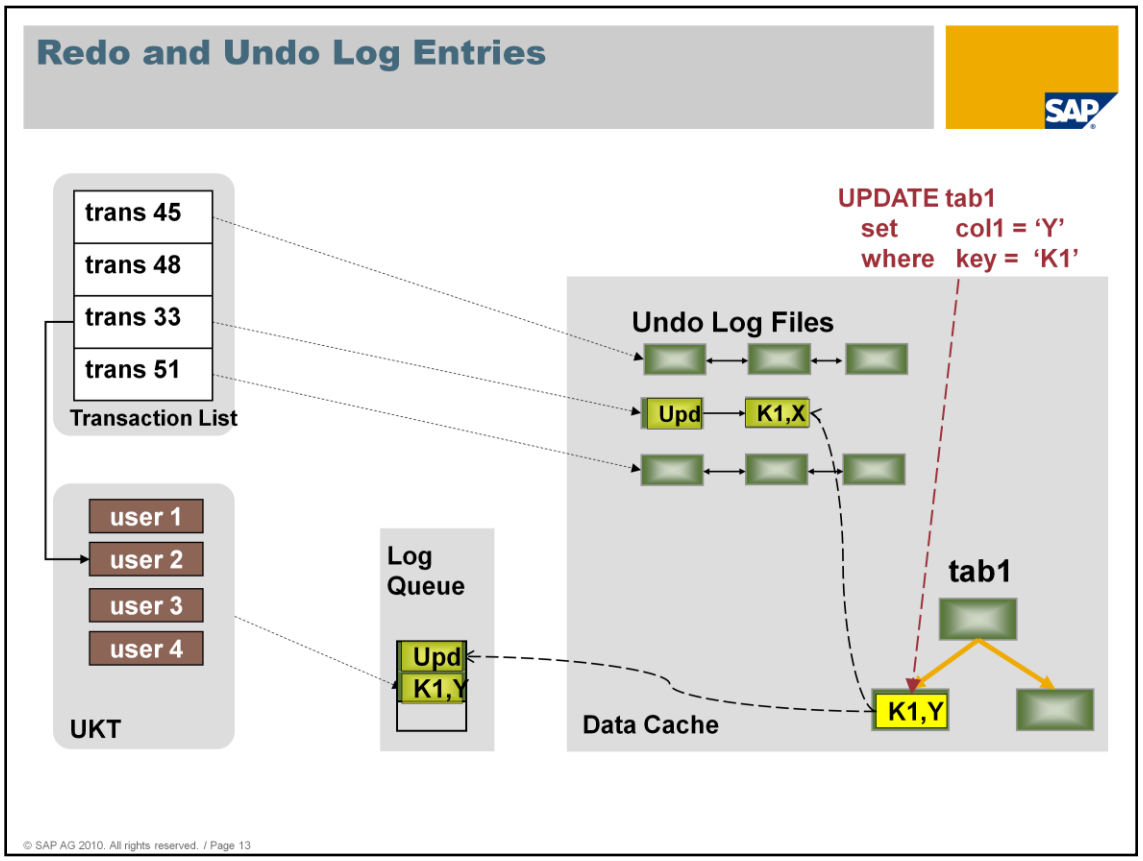
Log full state may occur for each partition. Some user tasks may be in a waiting state because of a filled log volume of their partition while other user tasks working with another partition can still perform changing operations.

The DBM extends the log area by one volume at a time. If you want to use multiple partitions specify the new volumes individually with their corresponding partition:

```
db_addvolume LOG DISKL001_2 F 18750 PARTITION 1
db_addvolume LOG DISKL002_2 F 18750 PARTITION 2
```

Transaction Management

CONNECT:	➢ Start of a MaxDB session ➢ Implicit start of a transaction ➢ No entry in log volume
COMMIT:	➢ Successful end of a transaction ➢ Implicit start of a new transaction ➢ Entry in log volume (only after changes)
ROLLBACK:	➢ Rollback of a transaction ➢ Implicit start of a new transaction ➢ Entry in log volume (only after changes)
COMMIT RELEASE:	➢ Successful end of a transaction ➢ End of the MaxDB session ➢ Entry in log volume (only after changes)
ROLLBACK RELEASE:	➢ Rollback of a transaction ➢ End of the MaxDB session ➢ Entry in log volume (only after changes)



MaxDB administers redo and undo logs separately.

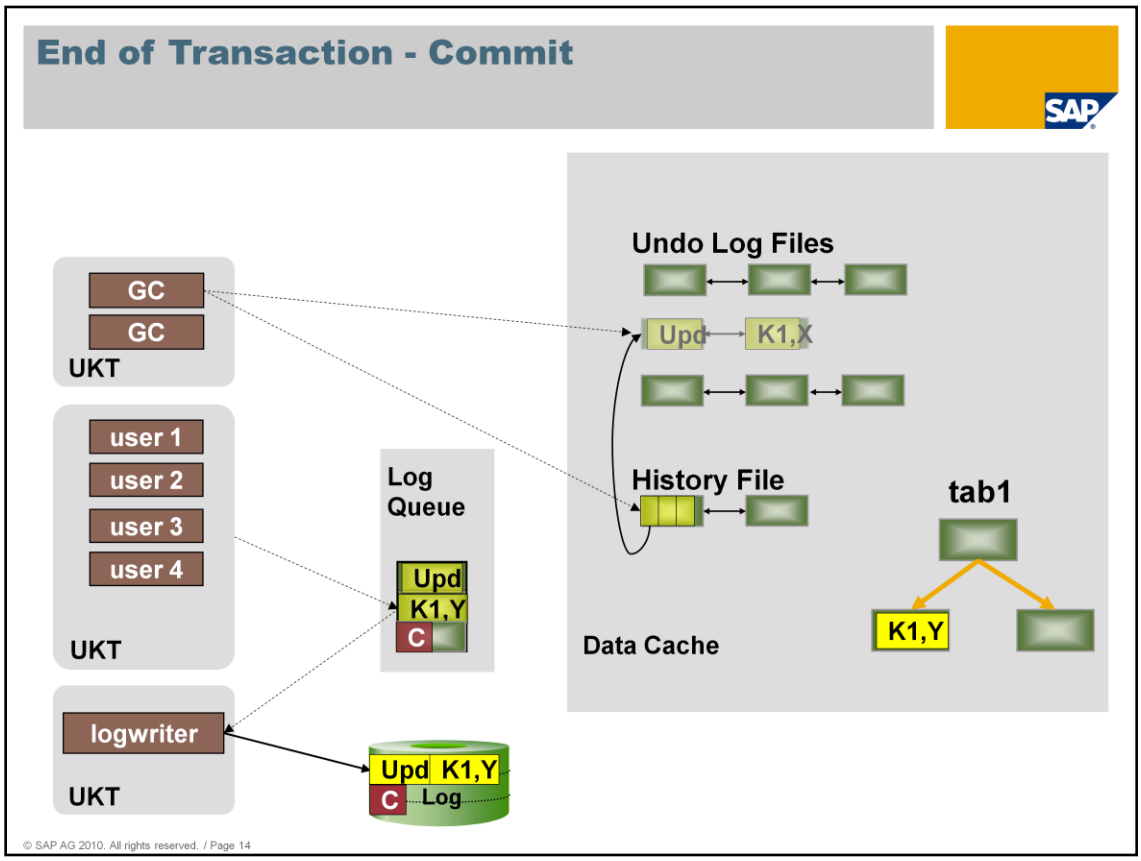
In the present example, user session 2 is working in transaction 33. It is updating a record in the table "tab1".

Transaction 33 has a link to an undo log file. The user enters the field values that were valid before the update into the undo log file with the action update and the primary key of the record. Each transaction uses its own undo log file. This prevents collisions from taking place while accessing undo log files.

The user copies the new field values together with the action and the primary key of the record into log queue.

Undo log entries enable you to roll back a transaction.

The redo log entries enable the complete recovery of data following a crash or disk errors.



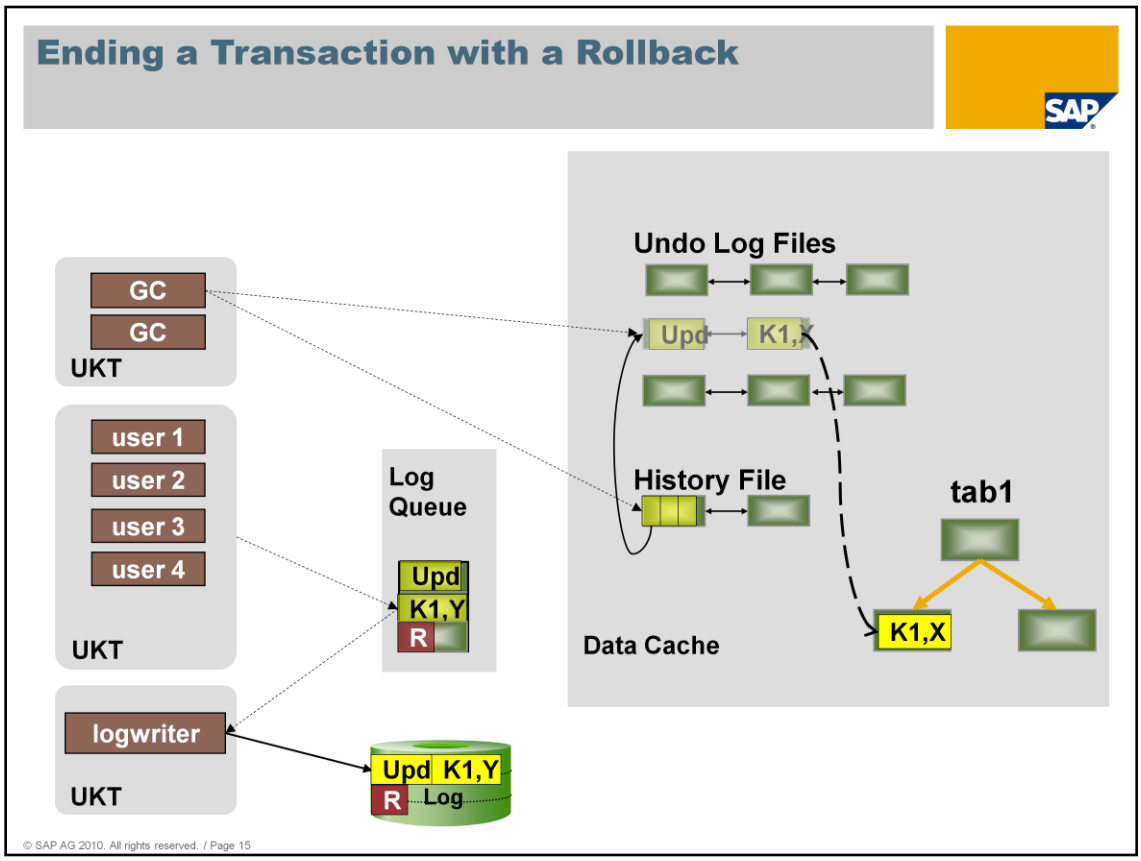
In the present case, the transaction is closed with a commit.

The user enters the commit in the log queue and activates the log writer.

The log writer reads all pages that contain entries of the transaction and were not yet written from the log queue and writes them to the log volume.

The user deletes the undo log file when the log writer has confirmed the successful write operation for the commit.

Garbage Collectors regularly check the history file every few seconds. They delete the undo log files found there. Thus the user task is disburdened by the delete operation of the undo log entries. The database can then confirm the Commit to the application more quickly.



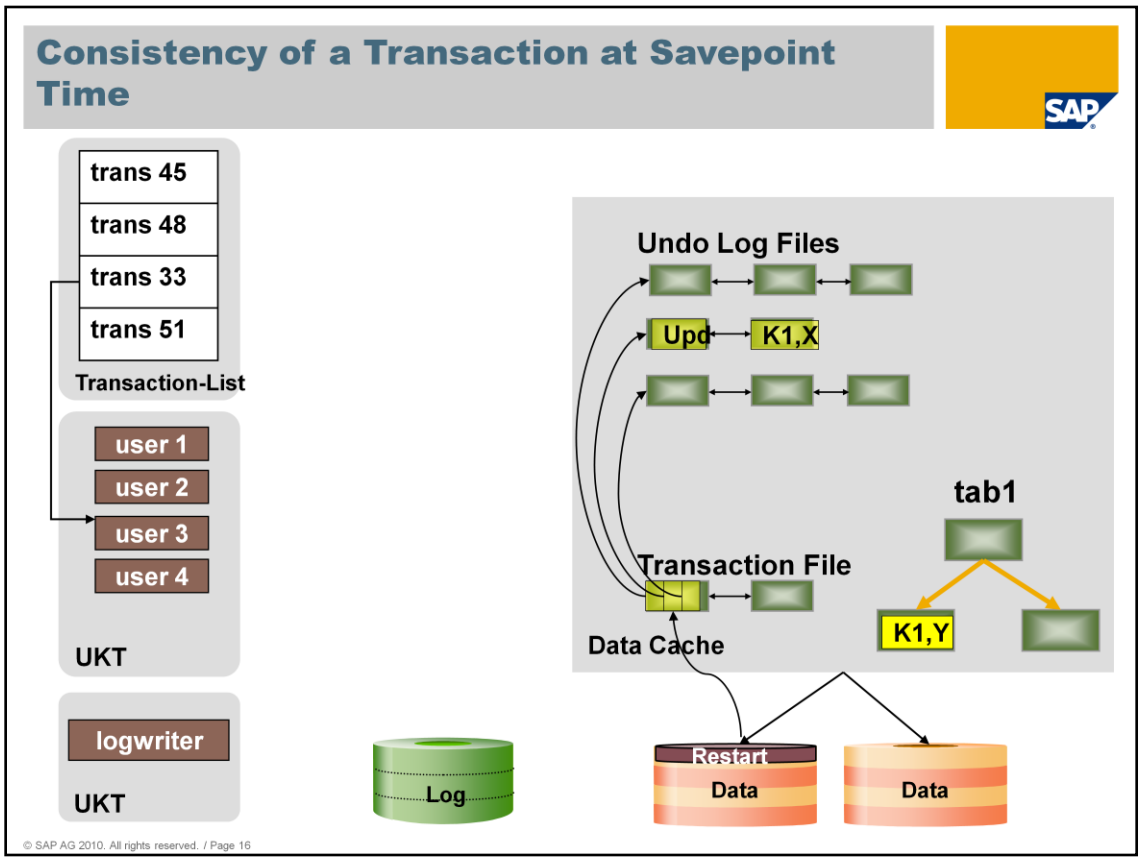
In a rollback, the user first retracts all the changes to the table data that are listed in the undo log file.

The user enters the rollback in the log queue and activates the log writer.

The log writer reads all pages that contain entries of the transaction and were not yet written from the log queue and writes them to the log volume.

The user deletes the undo log file when the log writer has confirmed the write operation to the log volume.

Garbage Collectors regularly check the history file every few seconds. They delete the undo log files found there. Thus the user task is disburdened by the delete operation of the undo log entries. The database can then confirm the Rollback to the application more quickly.



Undo log files consist of one or more permanent data pages. A savepoint handles them just like data pages with table and index information; in other words, they are written to the data volumes.

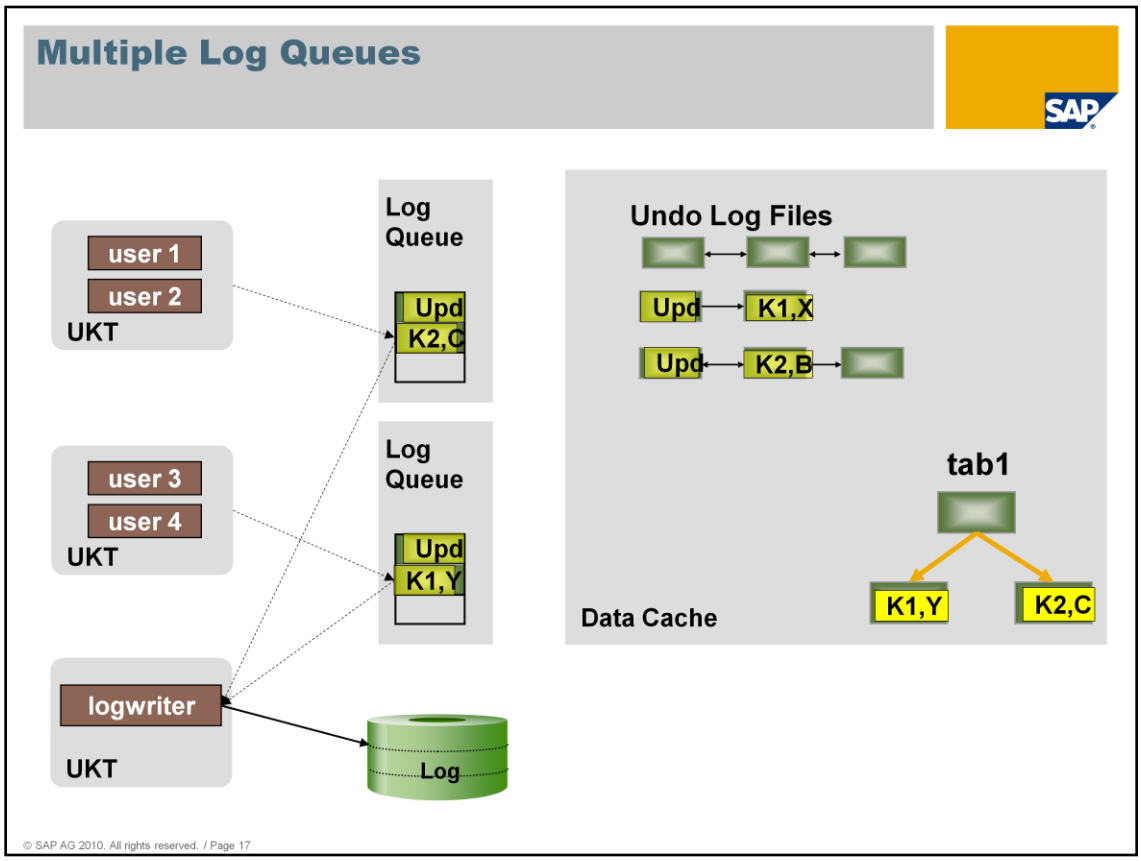
History administration notes the first page of each undo log file in the history file. The first page of the history file is recorded in the restart page at the end of the savepoint. So a restart can start at the last savepoint and find the actions of all open transactions.

With a savepoint, the database writes a status in the volumes that can be reassumed without use of the log.

If the savepoint is created for a data backup, this data backup can be imported into another instance as a system copy. The data backup also contains the undo log files for open transactions.

A restart of the system copy works even though no corresponding logs have been provided. In this case, the restart undoes all transactions that were open at the time of the savepoint. The requisite information is available in the form of the undo log files.

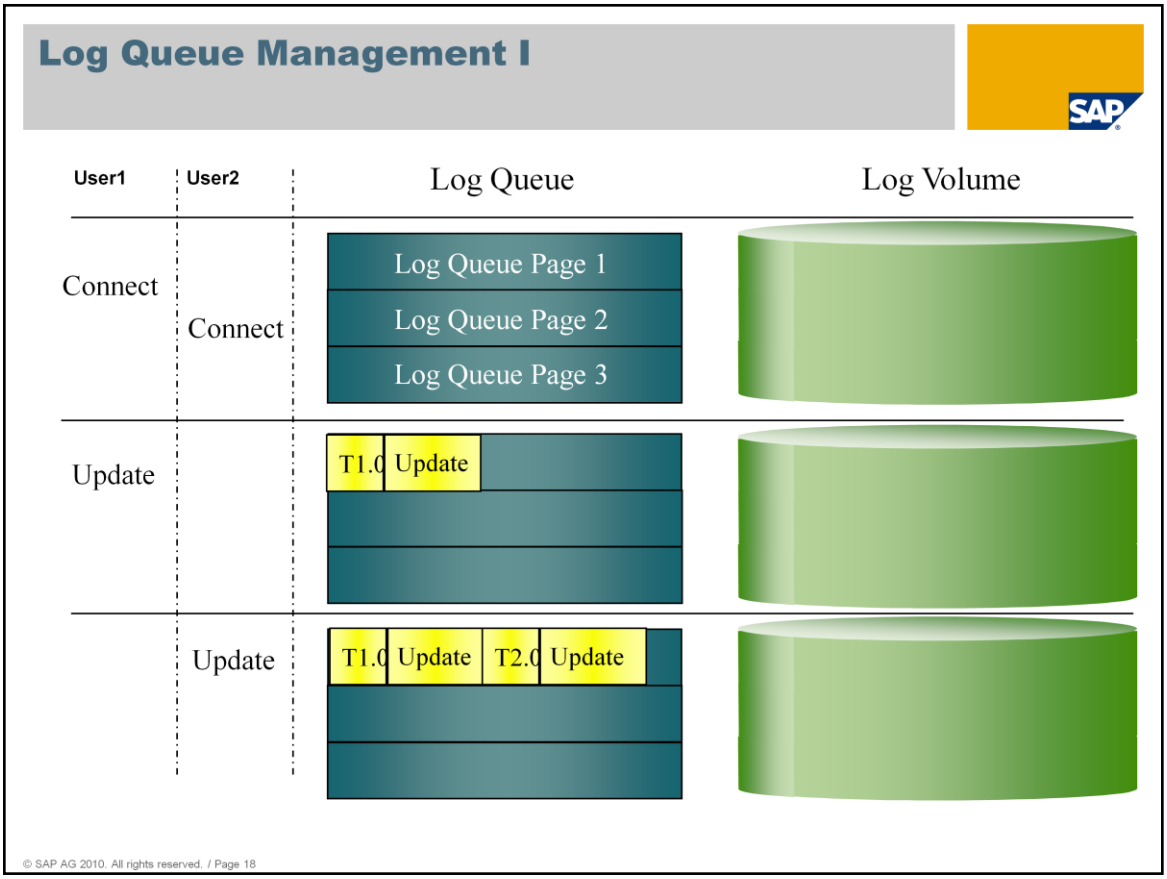
Before restarting the system copy, after restoring the data backup you can also import log backups.



SAP MaxDB supports the use of multiple log queues. The database parameter LogQueues determines the number of log queues.

In the standard, the value for LogQueues is equivalent to the value for MaxCPUs. Each UKT with user tasks writes to its own log queue. This prevents collisions at the log queues.

The database still works with one logwriter, which imports the log pages from the log queues and writes them to the log area.

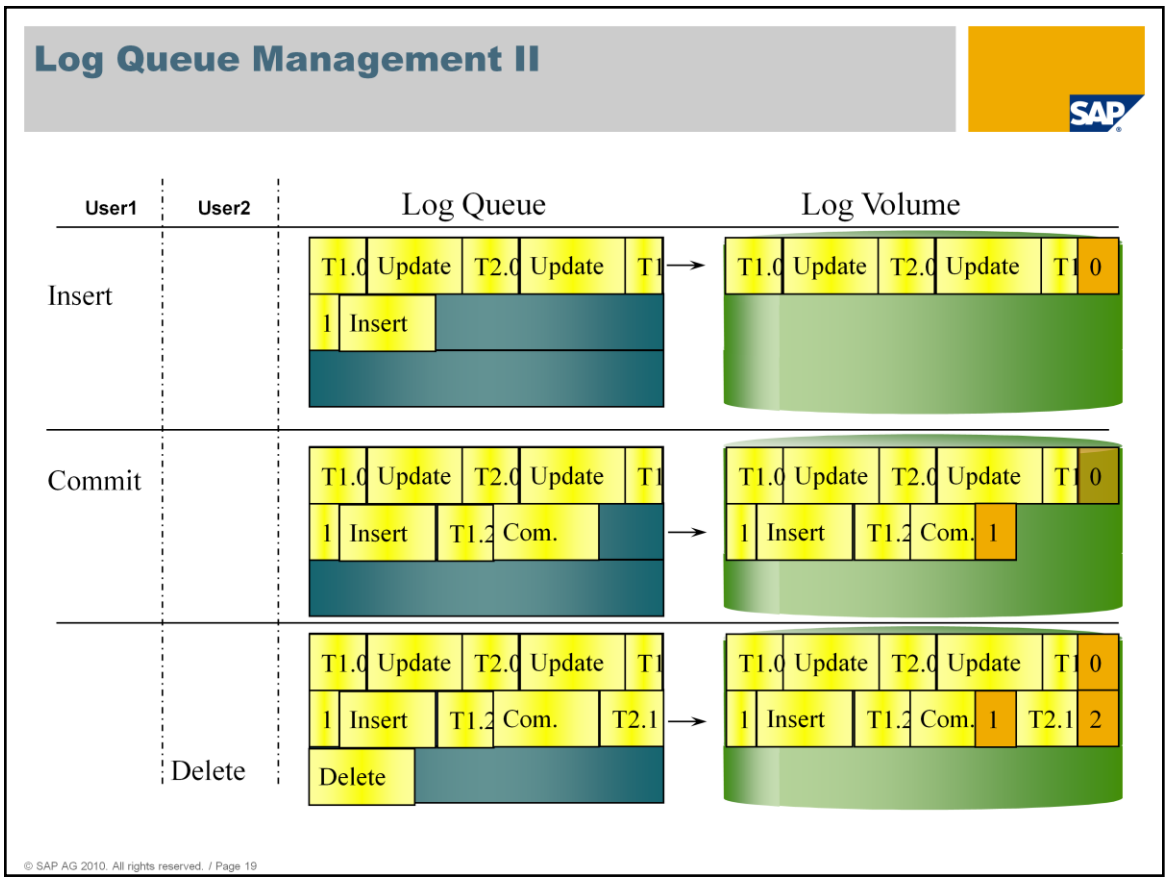


In this example the log queue has a size of 3 pages. Users write the redo log entries in the log queue, but not directly to the log volume.

Users 1 and 2 log on to the database. Only redo log entries are written in the log queue, so the connect does not need to be written in the log.

User 1 executes an UPDATE statement in the database. An update changes data in the database, so a redo log entry is made for this statement in the log.

User 2 then executes an UPDATE statement. The entry in the log queue gets the ID T2.0.



The INSERT executed by user 1 fills the log queue page. The first part of the redo log entry is written in the first page of the log queue page. The second part is written in the next log queue page. The user now instructs the log writer to copy the log page to the log volume.

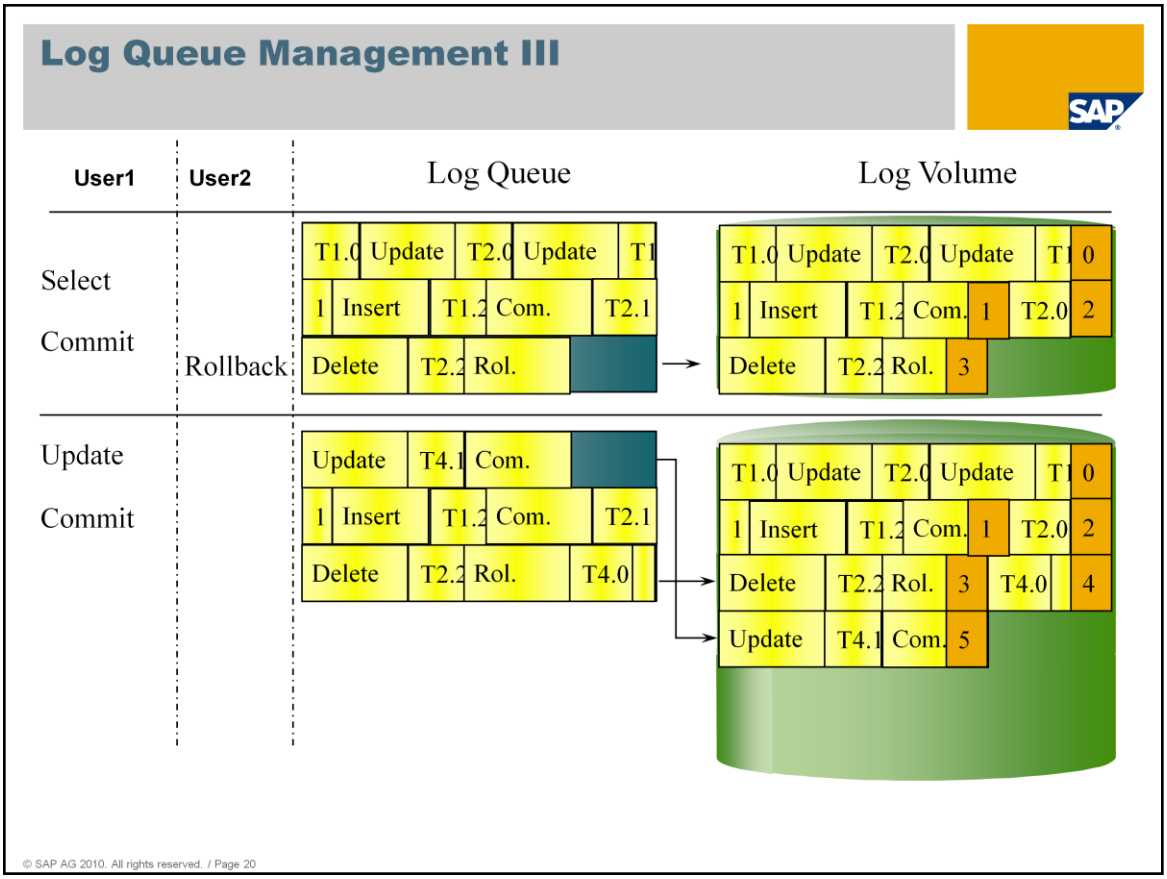
An I/O sequence is assigned for each write I/O in the log volume. This I/O sequence serves to synchronize the parallel restore log or restart. When the first log page is written, it is assigned the I/O sequence 0.

User 1 ends the transaction with a commit. The transaction can only be confirmed as completed and ready for application when the commit is in the log volume. The log writer writes the incompletely-filled log page to the log volume and assigns a new I/O sequence.

User 2 fills the second log page with the redo log entry for a delete. The log writer writes the page to the log volume. User 2 can continue to work and does not have to wait until the page is in the volume.

If multiple log queues are used, the logwriter will not overwrite blocks in the log. New entries always will be put to the next block. This leads to a certain higher log consumption.

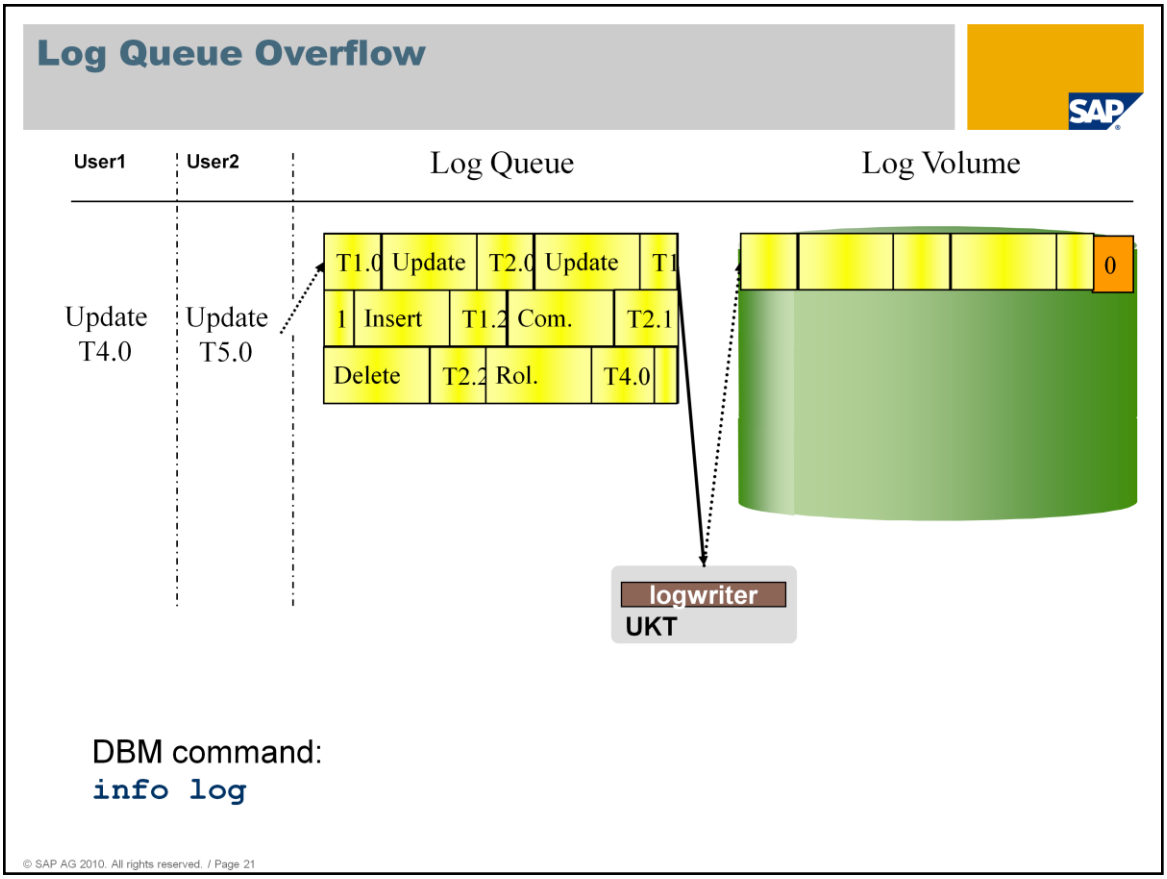
If multiple log partitions are used the I/O sequence numbers may be distributed to different log volumes. Redo Log recomposes the order of sequences in a consecutive manner when reading from the log partitions.



User 1 selects only data for the duration of the transaction. The transaction 3 makes no changes. No log entries are written for the selects and the subsequent commit.

User 2 ends the transaction with a rollback. When the transaction is rolled back, the undo log entries are read from the undo log files in the data area. When all undo log entries have been rolled back, the redo log entry for the rollback is written in the log queue. The user waits until this entry is in the log volume.

User 1 makes changes and completely fills the log queue. Further entries are now written in the first page of the log queue. With the commit, this page is also copied to the log volume. As the log area is generally larger than the log queue, writing can continue there.



If the log writer cannot copy the log pages in the log queue fast enough, the log queue can fill up with pages that have not yet been written. This effect is known as log queue overflow.

If this happens, all users who want to perform changes have to wait. Thus this situation can be very critical for performance.

In most cases, log queue overflows occur due to slow write I/Os for the log volumes. Mostly you will get less overflows if the I/O is accelerated. In high workload situations the configuration of additional log partitions may reduce the number of overflows. Sometimes it might be necessary to enlarge the log queue.

The DBM command "info log" displays the number of log queue overflows since the last restart of the instance.

Log Queue: Who has to wait?

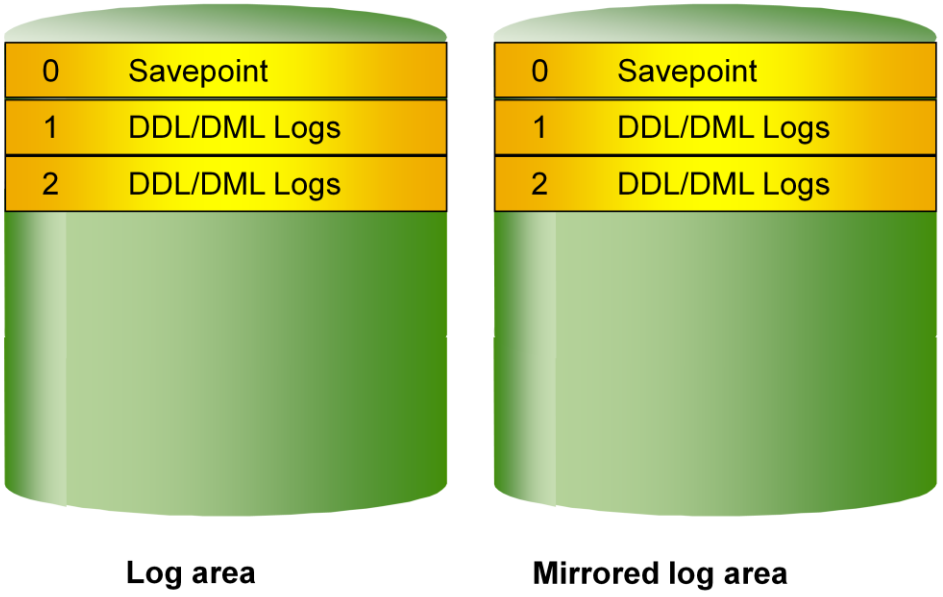
Copying entries from the log queue to the log volume is initiated by:

Condition	Transactions	I/O Wait
Log Page full	After copy the user process can immediately write to the next log queue page.	No
Commit / Rollback	The ending transaction has to wait until all log entries belonging to it have been written to disk.	Yes
Savepoint	None of the transactions has to wait.	No

The copying of the log queue to the log volume is triggered by:

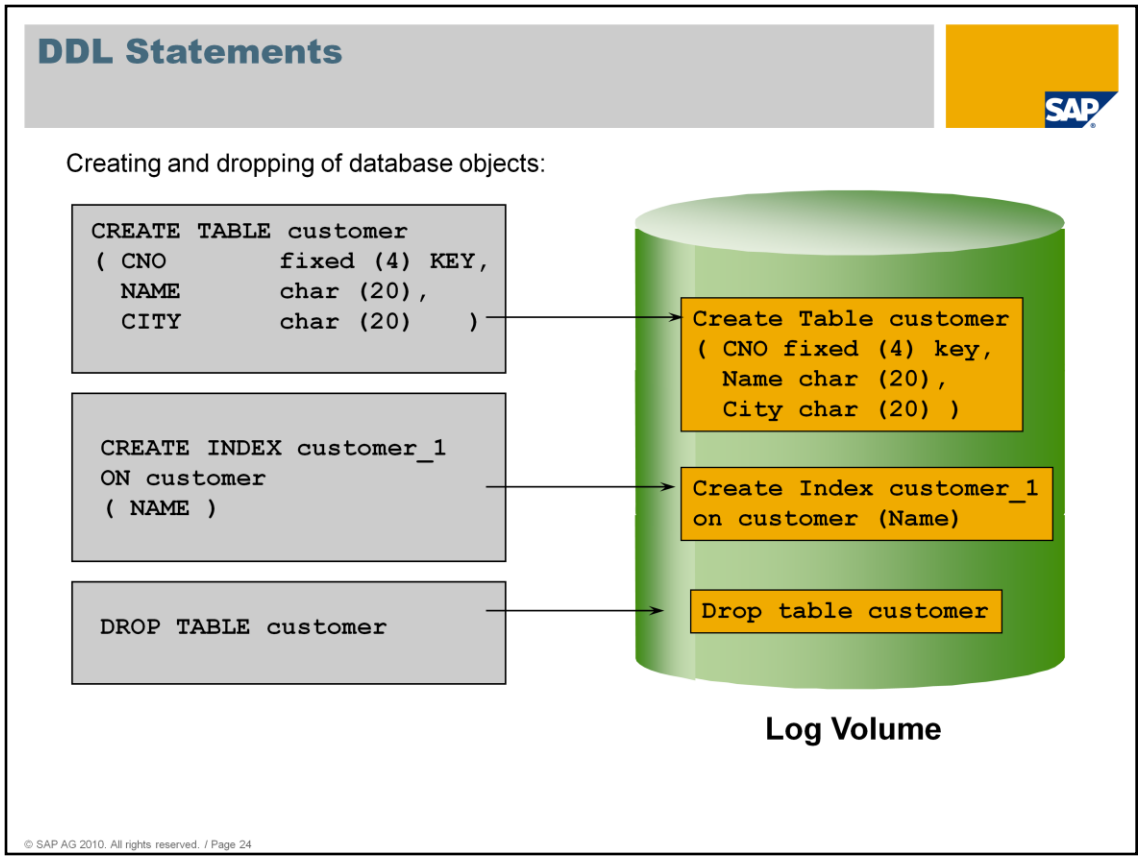
- A full log queue page
  The user transaction can begin to write the subsequent log page as soon as the entry is in the log queue. It does not have to wait until the log writer has written the entry in the log queue to the log volume.
- A commit or rollback
  The transaction has to wait until all corresponding entries have been written to the log volume.
- A savepoint
  User transactions can continue working. They do not have to wait until the savepoint entry has been written to the log volume.

Contents of the Log Volume



In a new installation of the database, the log volumes are formatted. The DBM command db_activate without option RECOVER launches the first restart of the database. A savepoint is also written. Thereafter, the database is ready for use and all database components can connect to the database.

All changes by data definition and data manipulation commands are written in the log.

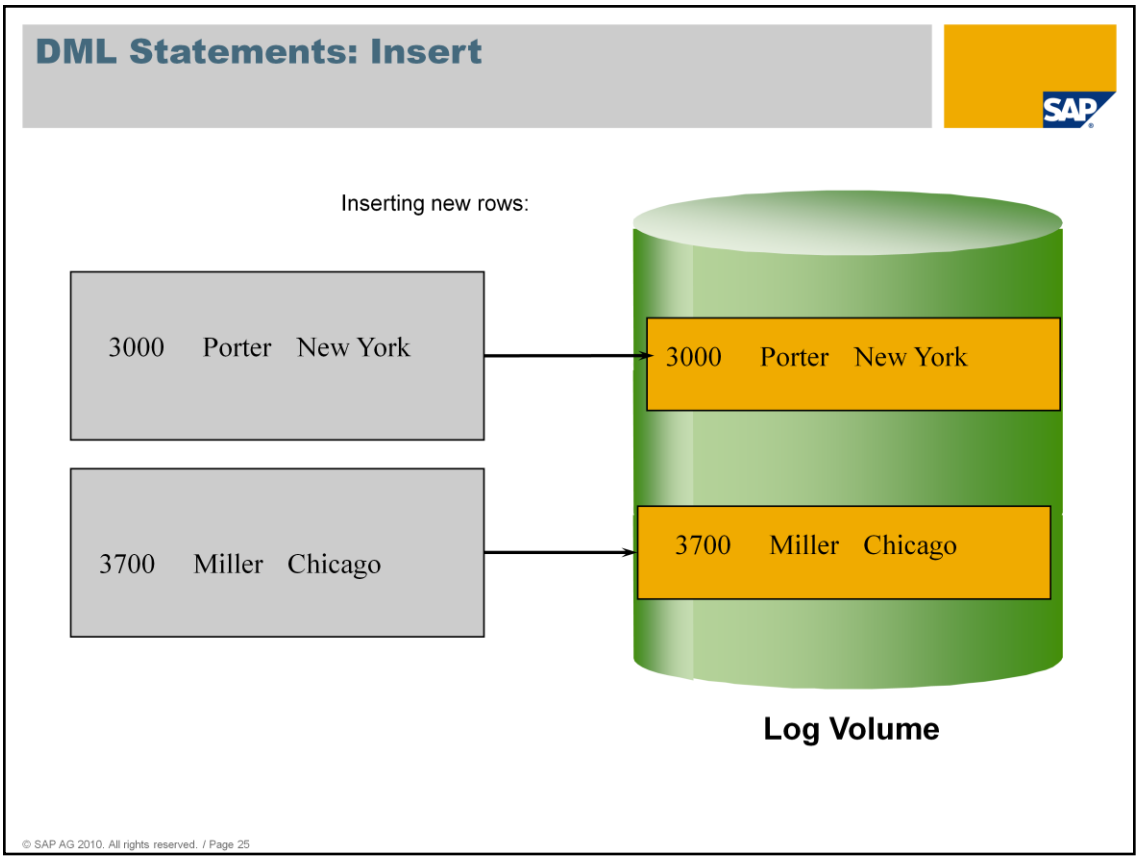


Creation and deletion of database objects:

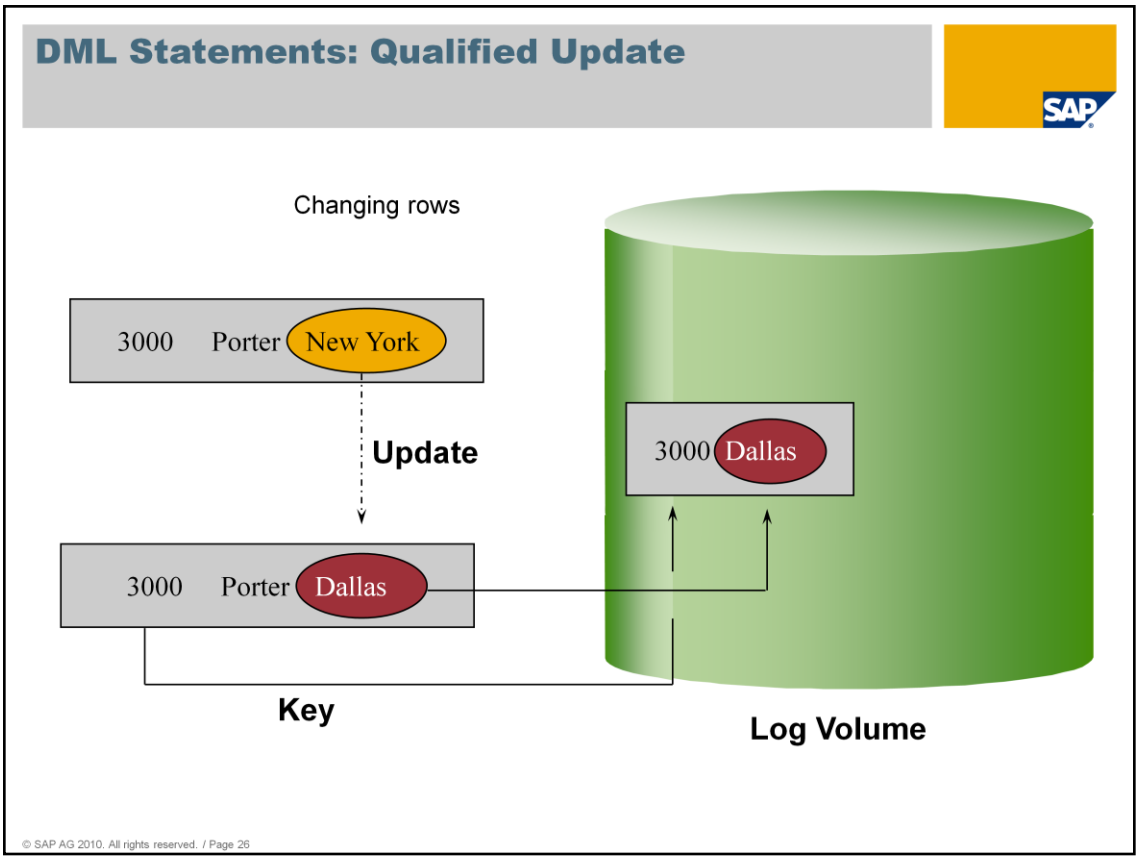
If a new object is created in the database, the CREATE statement is written to the log volume as a redo log entry. Moreover, the kernel writes an undo log entry to the undo log file of the corresponding transaction.

If an index is created in the database, this statement, too, is written to the log. The creation of an index also triggers a savepoint.

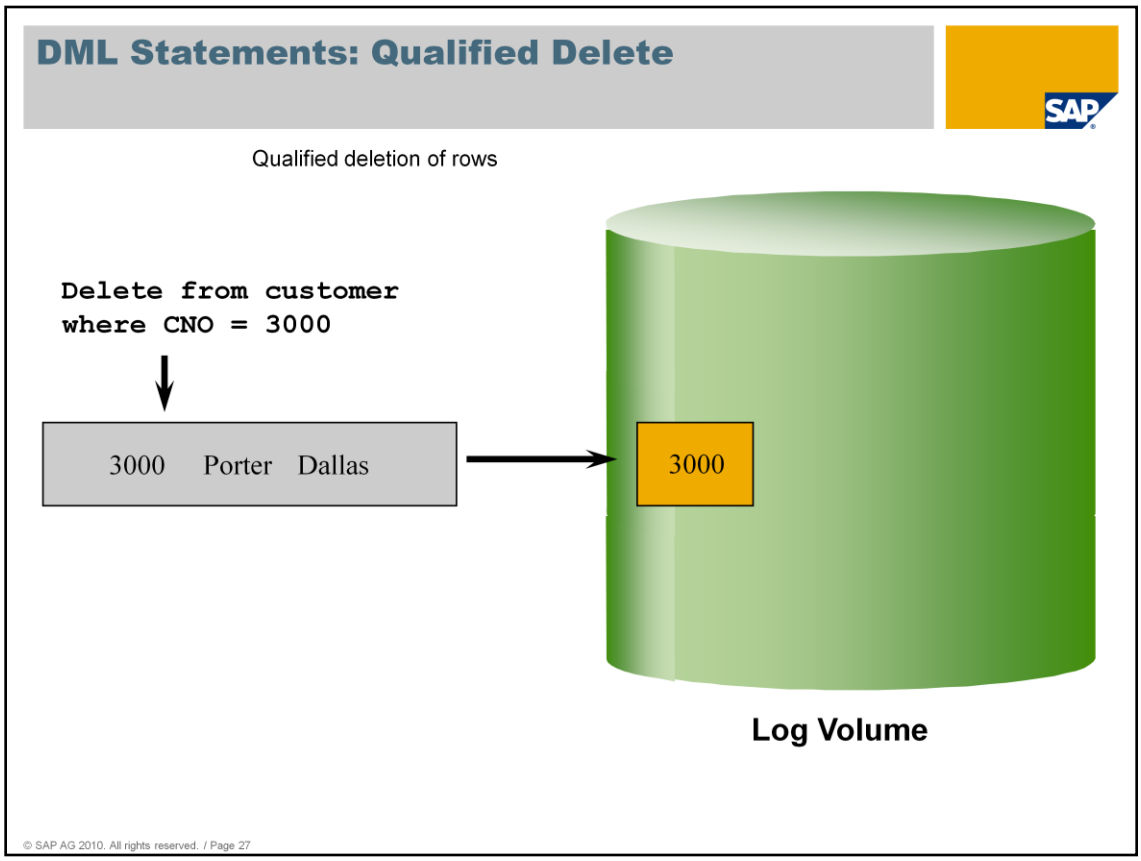
The statements are not written to the log in plain text, but rather in the form of stack code, which is generated by the SQL manager.



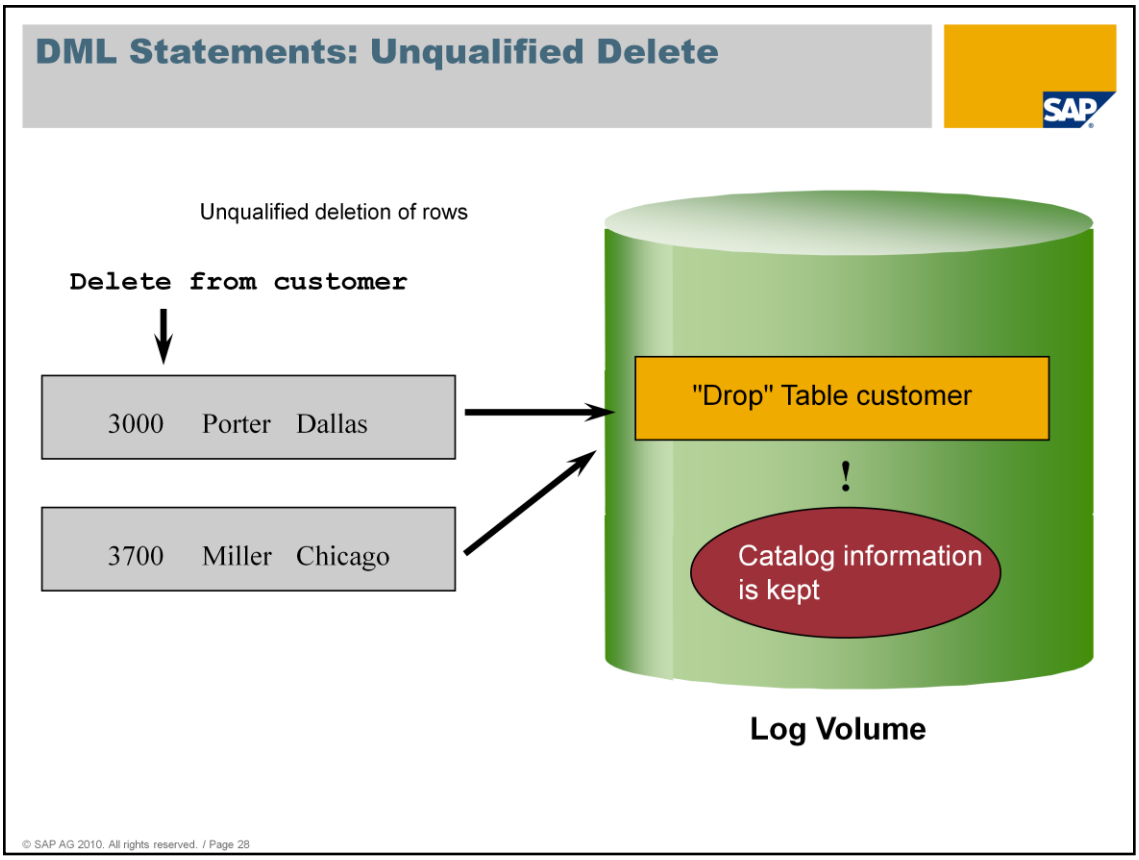
If new records are inserted into a table, the entire new record is written in the log.



If existing records are changed in the database, the key of the record to be changed is written in the log with the changed field values. For variable-length fields, the length byte of the field is also stored.



For qualified deletion of records, only the primary keys of and the table ID of the records to be deleted are written in the log. The other field values are not needed for a redo.



For unqualified deletion, the records to be deleted are not written in the log.

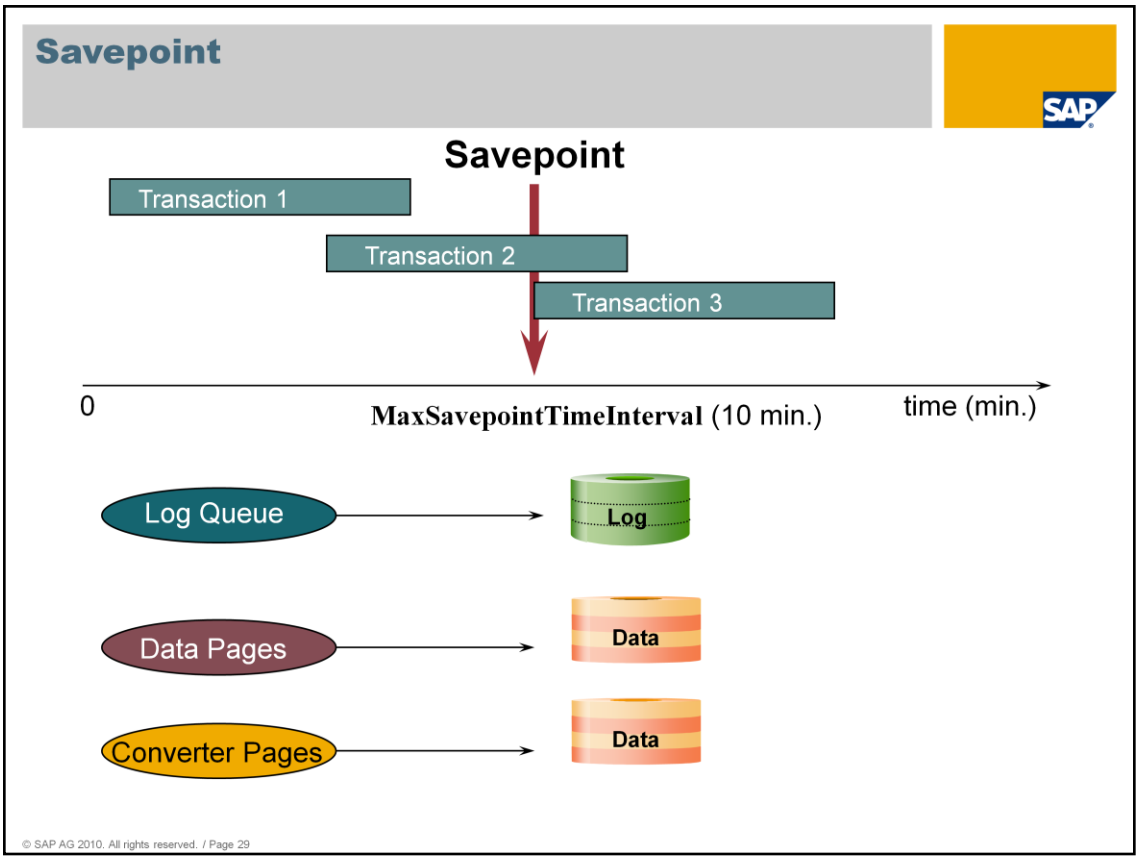
The deletion of all the records in a table is handled like a DROP Table, that is, a ‘DROP TABLE’ statement, which triggers the deletion of the records, is written in the log. But the deletion of the data records does not occur physically; rather, the B* tree of the table is recopied. That way the data can be restored quickly in case of a rollback.

After the commit, the tree is deleted recursively. This action is executed by a server task in the background. The user does not have to wait until the B* tree is deleted.

The catalog information of the table is retained in a deletion.

Even for unqualified deletion of the table data, the JDBC interface expects the number of deleted records in the return. If TRUNCATE is used, the database kernel does not determine the number of records to be deleted.

Unqualified delete determines the number of records directly from the file directory and therefore does not need to count.



A savepoint speeds up database restarts. Savepoints are written asynchronously.

Savepoints are executed at regular time intervals, but they are also triggered by certain actions (for example CREATE INDEX) in the database. In the standard, a savepoint is started every 10 minutes. You can configure the time interval between savepoints with the parameter MaxSavepointTimeInterval. With this parameter, you can specify the minimum number of seconds that must elapse after a savepoint before a new savepoint is started. The parameter only affects time-controlled savepoints.

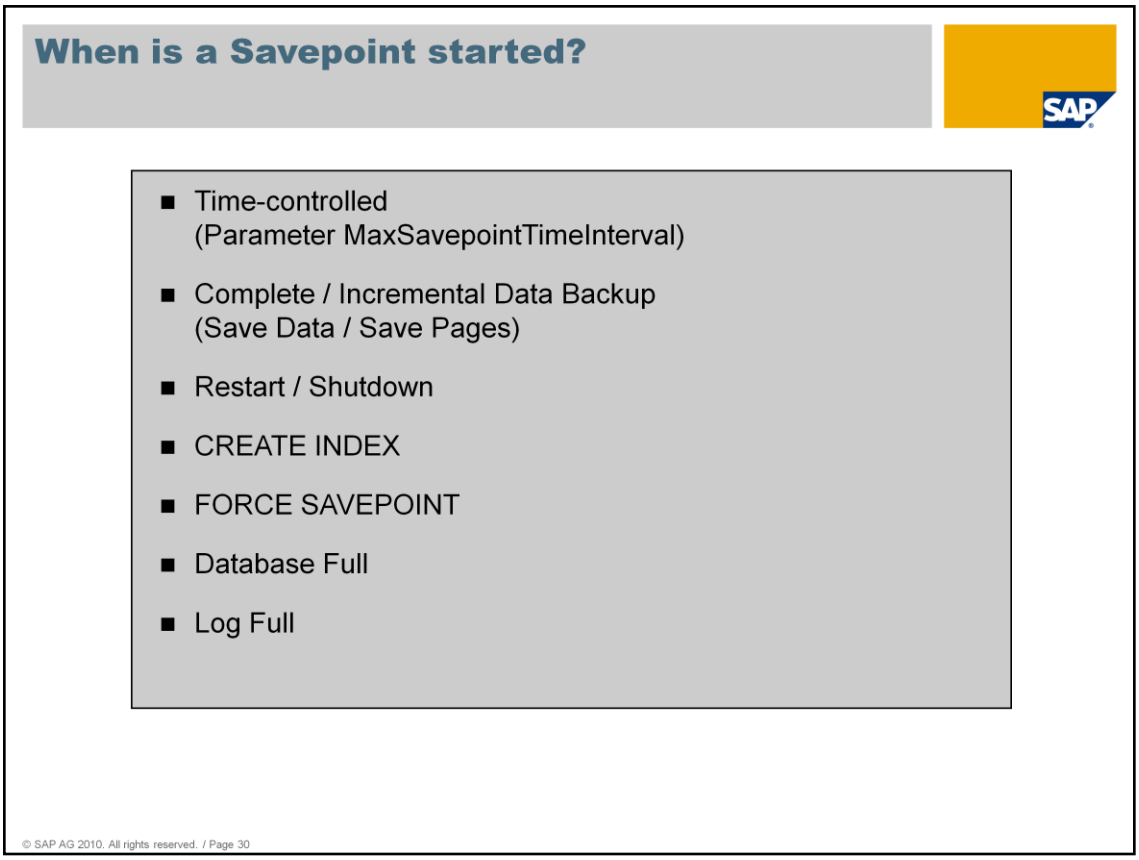
CAUTION: if you increase the value in MaxSavepointTimeInterval, you do reduce the number of savepoints and thus the workload, but this can also slow down the restart after a crash.

Too-frequent writing of savepoints causes the contents of the caches to be written too often with too little data, which hurts performance (I/O wait).

Time-controlled savepoints are not executed if no changes are made in the database.

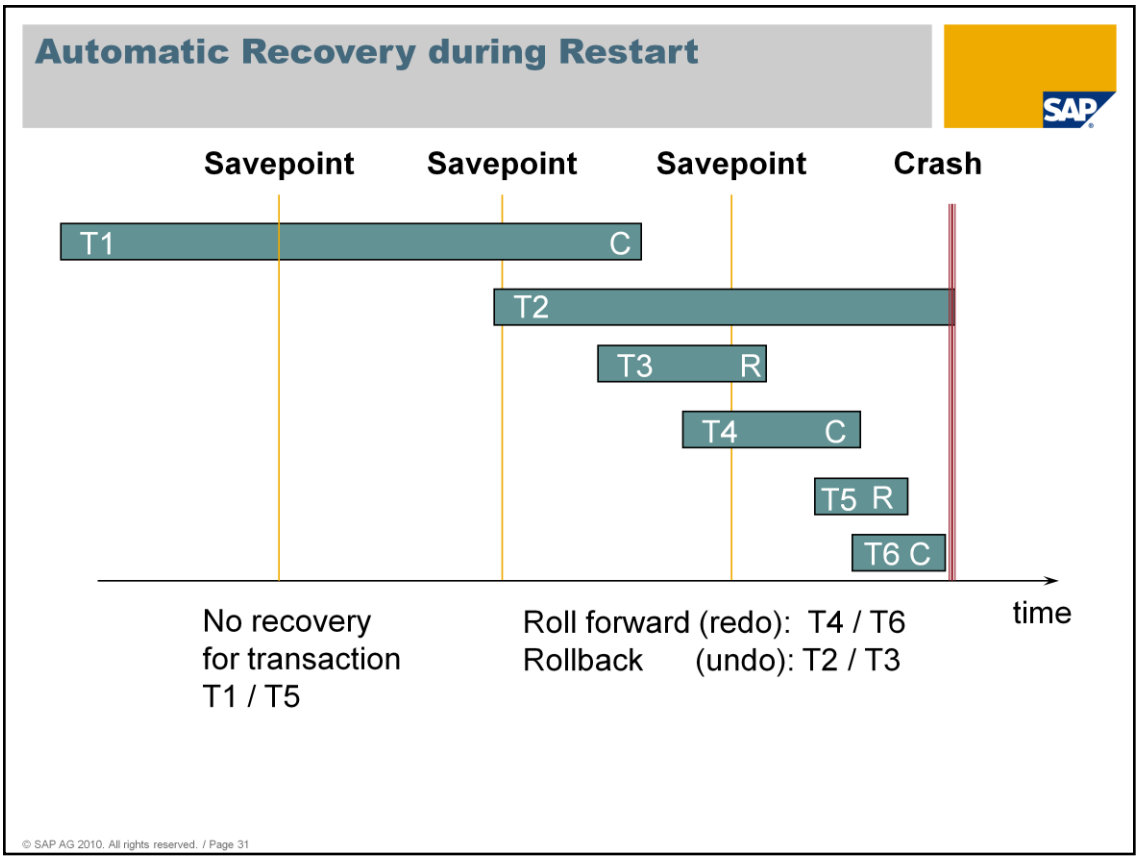
A savepoint is also written when a "CREATE INDEX" statement is sent by a transaction.

The savepoint triggers the writing of the log queue, data cache and converter cache to the volumes.



When is a savepoint requested?

- Savepoints are time-controlled. With the standard setting, a savepoint is started every 10 minutes.
- Data backups first start a savepoint and then back up all data covered by the savepoint. Changes to data during the backup are not included in the backup.
- When the database is shut down, the database kernel waits until a savepoint is carried out and all changed pages in the cache have been written to the volumes.
- Restart is ended with a savepoint.
- A savepoint is started at the end of a CREATE INDEX.
- You can start a savepoint manually with the SQL statement FORCE SAVEPOINT.
- When the kernel sees that the data area or the log is filling up, it starts a savepoint. This minimizes the restart time if the database is stopped with a full data or log area.



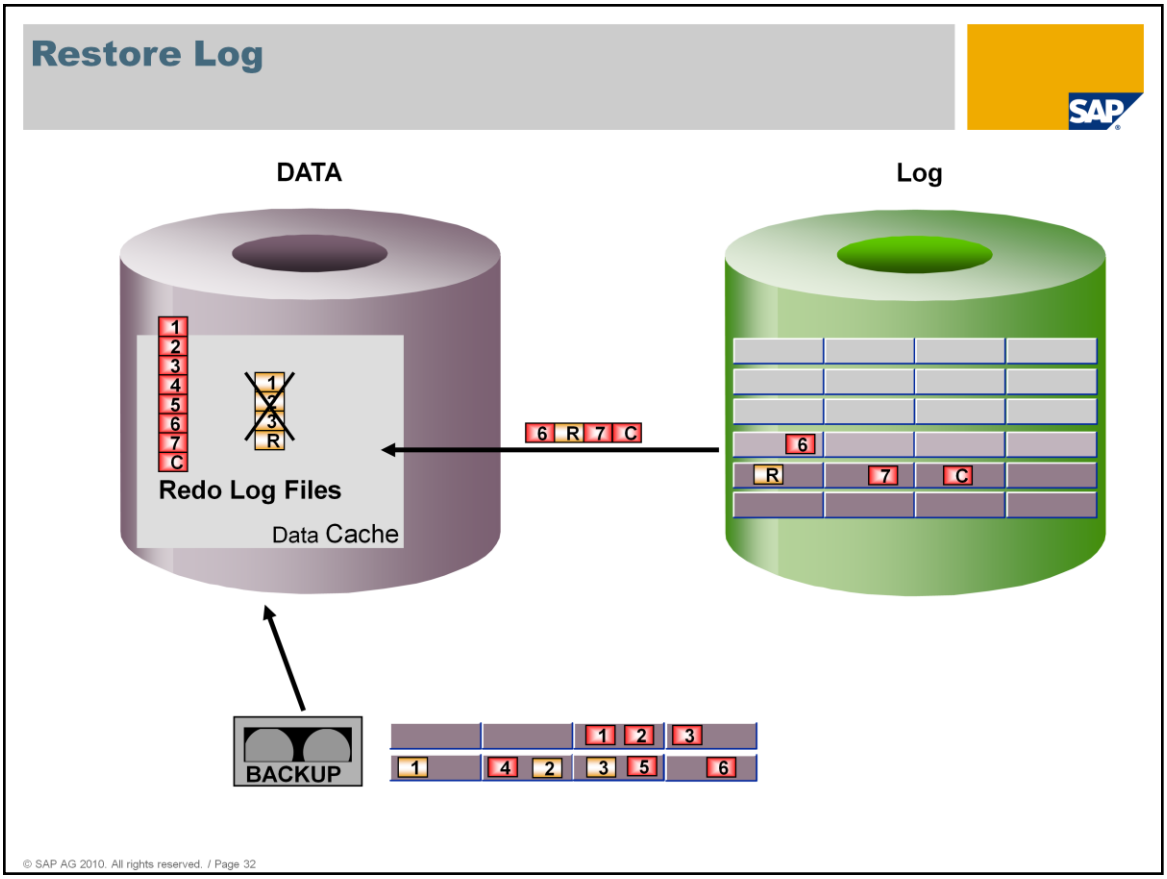
Recovery at restart

The restart implicitly redoes or undoes the transactions; to do this, it goes back to the last savepoint.

All transactions that were not complete at the time of the last savepoint are subject to a redo or undo. Completed transactions that were started after the last savepoint are also subject to a redo.

Example (see slide):

- Transactions 1 and 5 are not relevant for the restart. T1 is completely in the data area. T5 is not in the data area and was reset by a rollback.
- Transactions 2, 3 and 4 were not complete at the time of the last savepoint.
- T2 and T3 are rolled back from the savepoint. The changes of the transactions are read from the undo log files ➲ UNDO .
- T4 is redone starting at the savepoint. The changes are read from the log area ➲ REDO.
- Transaction T6 has to be completely redone. The changes are found in the redo log entries in the log area ➲ REDO.



Restore Log and restart create a redo log file in the data area for each transaction.

Only those redo log entries from the log backups that are no longer in the log volumes are copied.

A transaction is not redone until the redo log file has been completely generated.

Restore Log **without until** specification does not write to the log volume. Thus it is not important to back up the log area before the Restore.

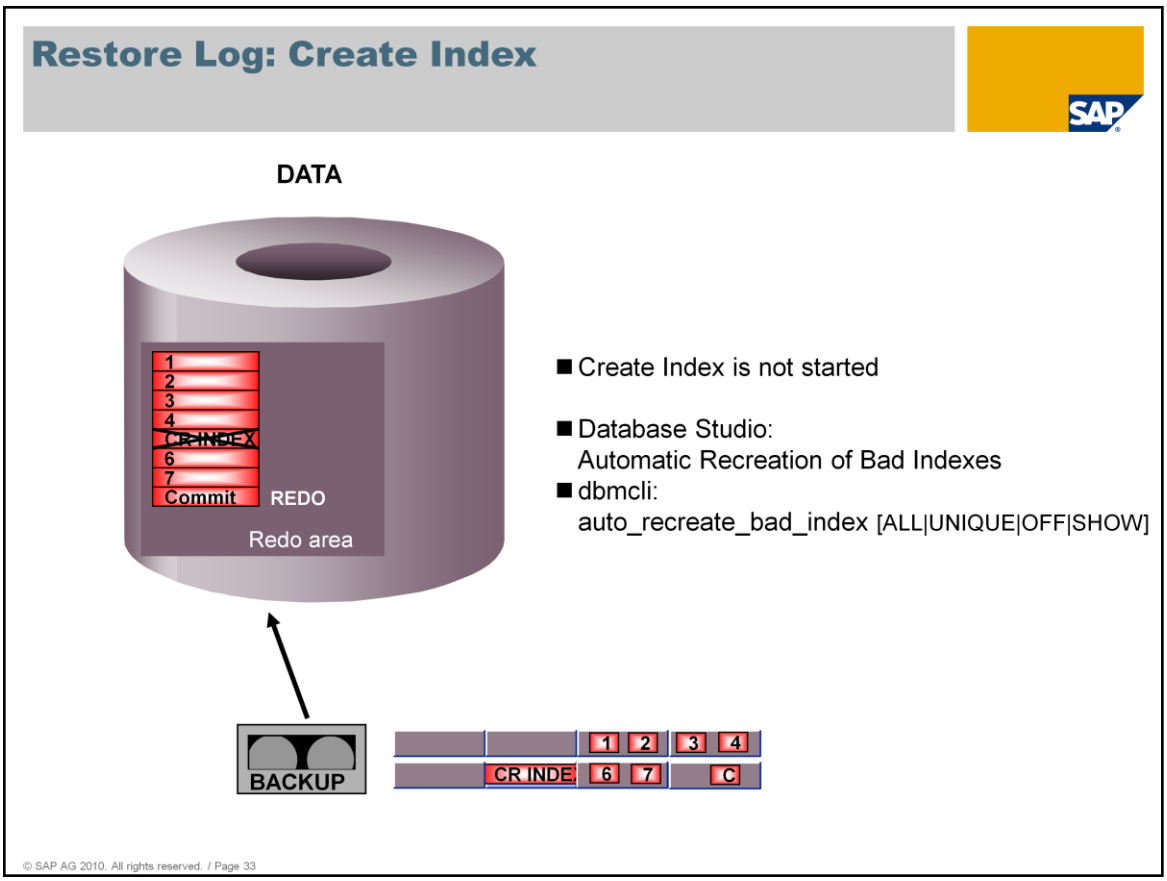
With Restore Log **with until** specification, after all redo log entries have been processed, a savepoint is written and the log is deleted from the until position onwards. This interrupts the log history. Therefore, in a production system,

- the log should be backed up before the Restore Log Until and
- a data backup should be generated once the Restore is completed.

Savepoints are written during the Restore Log. With these savepoints, the generated log files are also written to the log volumes. Following crashes, the last savepoint is the restart position. When you restart, start the Restore with the last log backup that was read.

The DBM command db_restartinfo in the ADMIN state displays the log page on which the Restore Log starts.

Whenever possible, SAP MaxDB redoes/undoes log entries in parallel.



CREATE INDEX statements are not repeated in the case of a redo.

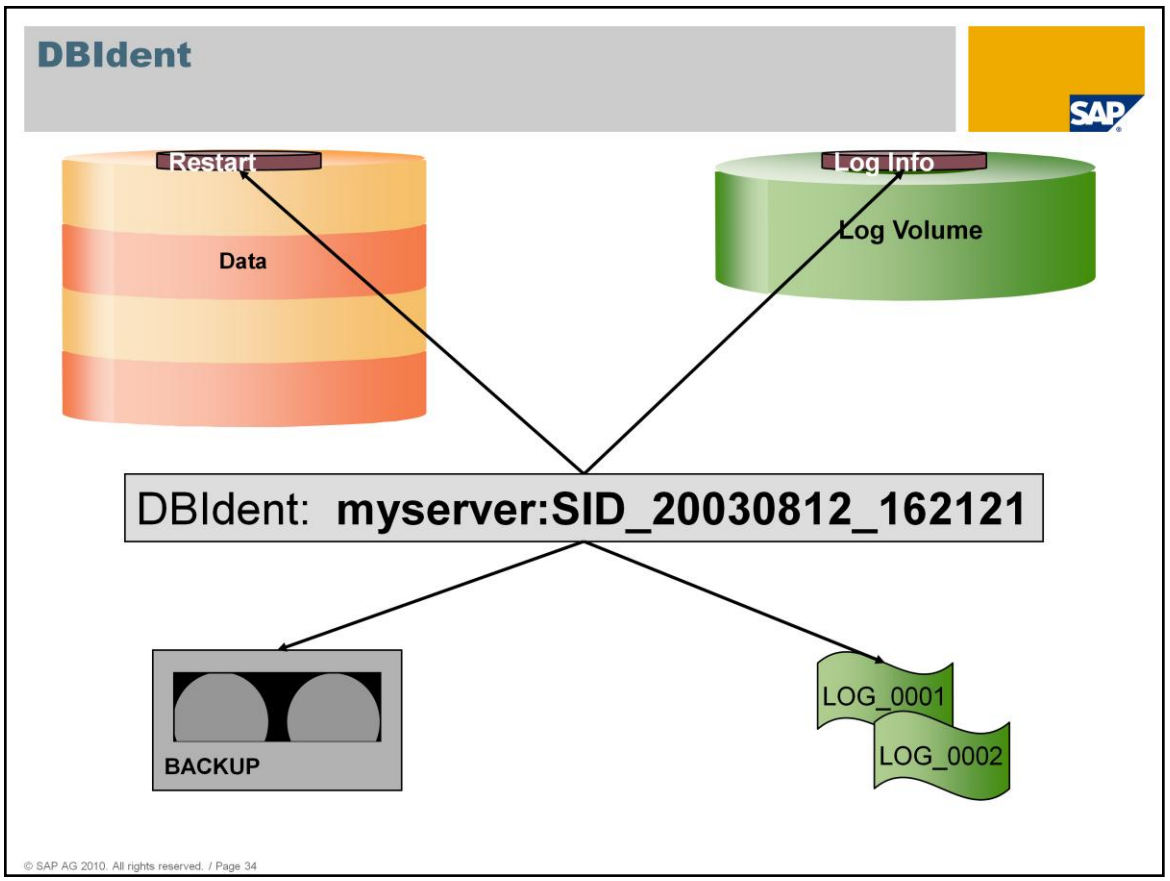
That can speed up RESTORE LOG considerably.

The indexes are recreated with RECREATE INDEX when the database is in the ADMIN state.

Version <= 7.7: When the parameter UseAutomaticBadIndexRecreation is set to YES, the corresponding indexes are automatically generated at the end of the restart/redo. Users can log on to the database when the indexes have been created.

Version >= 7.8: The database kernel sends an event to the DBM when it detects a corrupted index during online operation or at the end of the restart. The DBM treats the event according to the setting of auto_recreate_index:

- UNIQUE: Recreate Unique Indexes immediately
- ALL: Recreate all Bad Indexes immediately
- OFF: Do not automatically recreate Bad Indexes



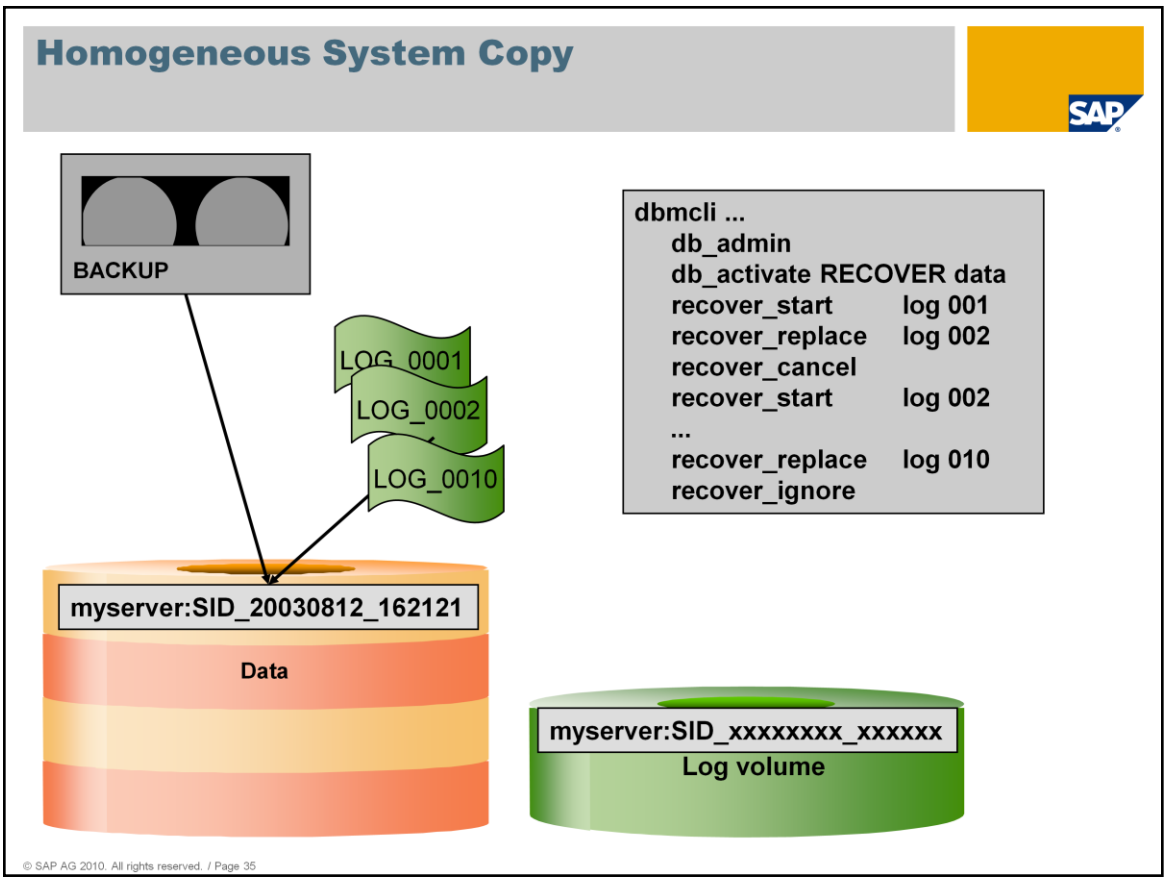
The log and data of an instance must always match. The data of one instance may not, for example, be mixed with the log of another instance during log recovery.

To avoid the occurrence of incompatibility between log and data, the database kernel writes the DBIdent in

- the restart record, which is located in the second block of the first data volume;
- the log info page, which is located in the second block of the first log volume;
- every data backup (Save Data / Save Pages); and
- every log backup.

The DBIdent contains the name of the database server and the database instance as well as a time stamp indicating when the backup history began. This time stamp is set with until specification, for example at the first restart and at the end of a Restore Log.

The names of the database server and the instance do not necessarily correspond to the current server and instance names. Database instances can be renamed without the DBIdent being changed (see OSS note 604680). Instances can change database servers in the HA cluster.



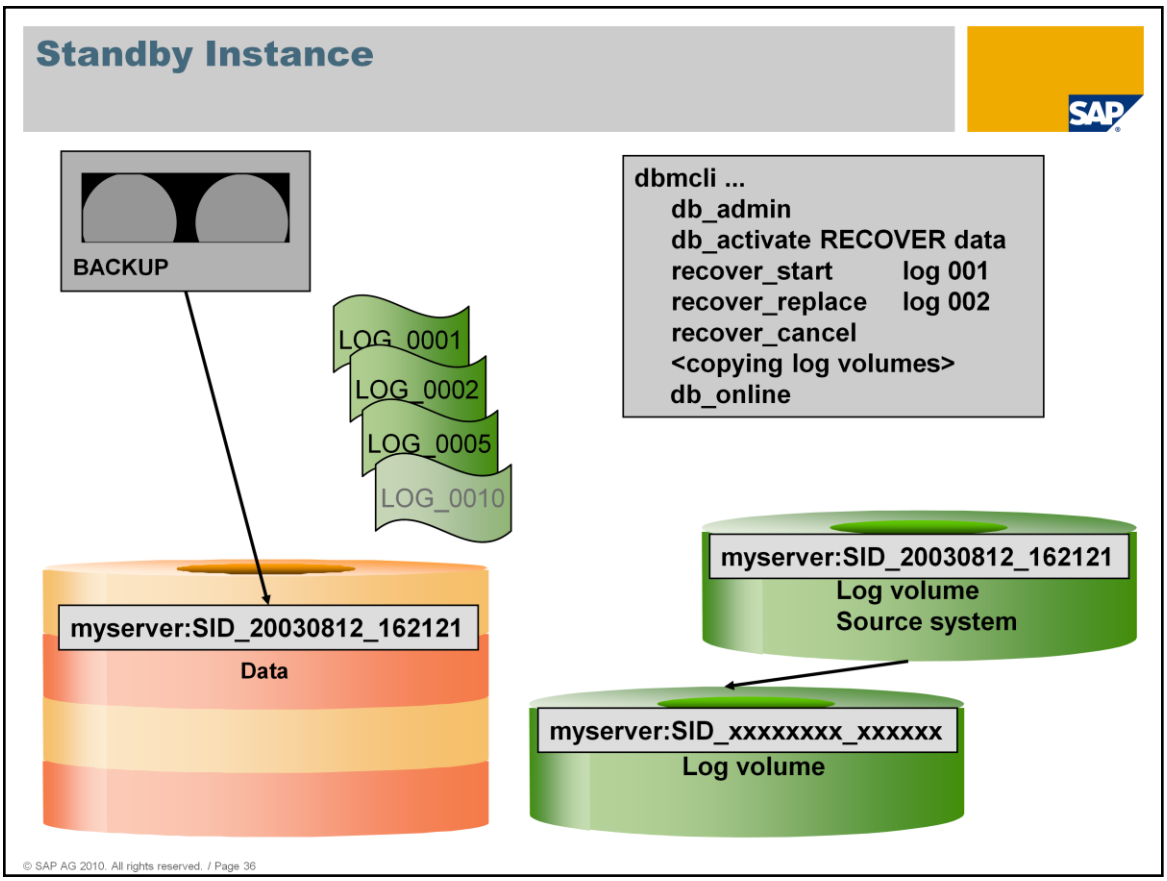
Homogeneous system copies are allowed if the processor types of the source and target servers are the same. For more information, see OSS note 129352.

Data backups contain the undo log entries of all open transactions. They can import a data backup into another instance per Restore and execute a restart. The restart ensures the consistency of the database by undoing the open transactions.

Note that in this case the restart only works if the DBIdent in the log matches the DBIdent in the data area or if the log was initialized with "db_activate RECOVER“. The "db_activate RECOVER“ statement does not fill the DBIdent. The restart is what sets the DBIdent in the data and log areas.

Restore Data enters the original DBIdent in the data area. This makes it possible to import log entries and log backups from the source system. The Restore Log process can be interrupted and resumed. In this case, the dbmcli command recover_ignore executes the restart and sets the new DBIdent.

If log entries from the source instance are redone in the target instance, the database version of both instances must be the same at build level.



MaxDB also offers the possibility of operating so-called standby databases. A standby instance is supplied with a data backup from an original instance. Subsequently the log backups from the original instance are imported into the standby instance.

If the log volumes of the original instance are still available before starting the standby instance, you can copy these log volumes to the log volumes of the standby instance. Then continue with recover_start. Once the redo log entries can be read from the log volumes, the log reader no longer reads from the log backups. The database is automatically set to the ONLINE operational state.

The DBIdent is retained. The log backup history is not interrupted.

Caution: The "db_activate RECOVER“ should not be used for a normal recovery. "db_activate RECOVER“ is used in the creation of a new instance. For a normal recovery, "db_activate RECOVERY" would unnecessarily prolong the recovery time. The current log in the log area is lost.

SAP MaxDB also supports hot standby environments.

Questions and Answers

Thank You!
Bye, Bye – And Remember Next Session

August 3, 2010	Session 11: Backup and Recovery

