NOTE APRIL 2008

SAP® table buffering

SAP® stores almost all data in tables. These tables are part of the underlying database. With that SAP® would need to execute a high number of database calls every time a corresponding information is required.

As part of a network in client-server environments this would be a time and resource consuming task with negative impact on system performance.

To reduce the corresponding workload SAP® has established a table buffer on every instance.

The data are then available through the buffer of the application server.

The direct database table access via network takes approximately 10-100 times longer than via local buffer.

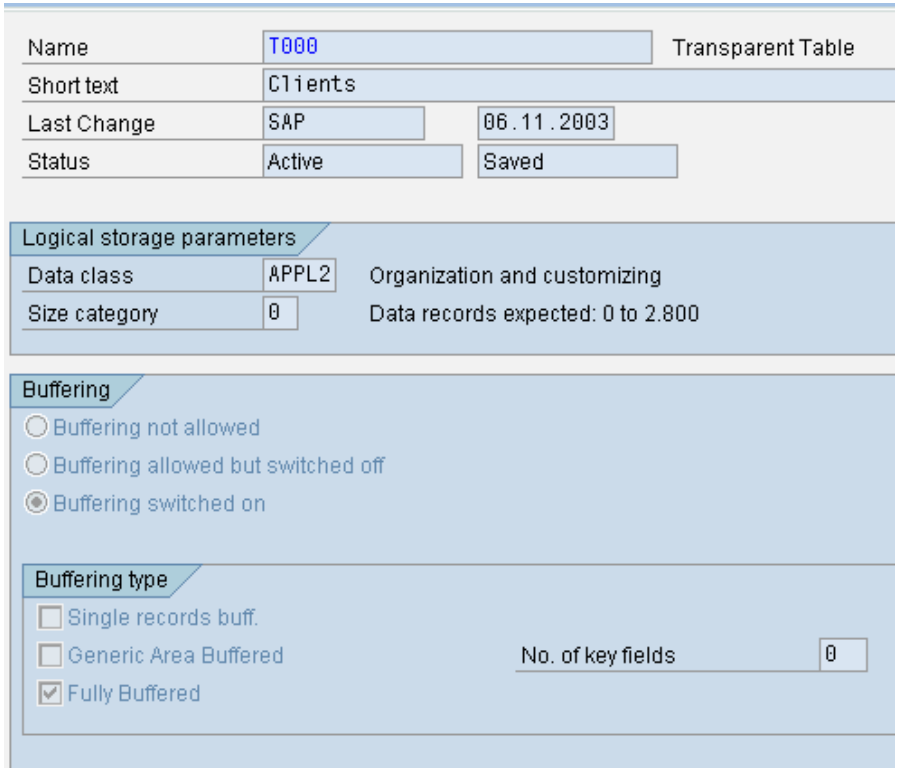
Currently there are three different buffer methodologies:

1. Full buffering
   When a table record is read, the table is completely loaded into the buffer.
2. Generic area buffering
   The generic key is part of the primary key of a table. When a table record is read, this buffer type loads all records that correspond with the generic key.
3. Single- record buffering
   The records that are read are loaded to the buffer.

With regard to the different buffer types, SAP® has two table buffers.

The buffer **TABL** contains the full and generic buffered tables, while the single-records are buffered in **TABLP**.

The buffer methodology is only available for transparent and pool tables - cluster tables are excluded – and is controlled via the technical table settings as part of transaction **SE13**.



- Buffering not allowed
- Buffering allowed, but switched off
- Buffering switched on

If the buffering is supposed to be active, the buffering type needs to be selected as well.

The advantage of buffering is quite obvious, but a valid question at this point is – how do the table contents stay up to date and synchronised in case of changes?

The answer is *buffer synchronisation*.

There are two types of buffer synchronisation, *synchronous* and *asynchronous*.

1. *synchronous*
   At a change the buffer of the local instance where the record was changed as well as the corresponding database entries are synchronised at the same time. The changes to buffers in local instances are centrally stored in the database table **DDLOG**. The other instances need to be updated as well. This is done by a second wave of synchronisation – the asynchronous synchronisation.
2. *asynchronous*
   The entries in the table **DDLOG** are checked by all application servers in a predefined frequency, controlled by the system parameter *rdisp/bufretime*. The parameter defines the waiting periods between two synchronisations in seconds and can be reviewed with the help of the transaction **SE38** and the report **RSPFPAR** or **RSPARAM** e.g.

   As soon as the instance identifies data that were changed as part of the local buffer, the corresponding entries [matching the buffer type] are set to invalid. At the next data access, the table information are directly read from the database. The table may then be loaded into the buffer again, but to avoid that a table is continuously reloaded to the buffer, a table is only loaded into the buffer after a certain waiting period.

A buffer reset can be performed by entering **$TAB** to the command line. Only when the buffers are definitely inconsistent this may be performed. The buffer reload can take several hours and may have a critical impact on the system performance. Access to this task needs to be restricted to members of the basis team only [authorization object S_ADMI_FCD value SYNC]. A manual synchronisation via **$SYNC** is hardly ever necessary, and should only be executed in exceptional, very special cases.

Buffering is not suitable for all tables. Usually tables with customizing entries or system tables are a good choice, as they are not modified with a high frequency.

If more than 1% of all table operations for a selected table are modifications, the benefit of buffering does not exceed the performance effort related to snychronisation.