SAP® MaxDB™
SQL Locks
Version 7.7

Heike Gursch
Werner Thesing

THE BEST-RUN BUSINESSES RUN SAP™

Overview

Transactions

Locking objects

Locking types

Locking conflict

Configuration

Lock escalations

Monitoring the locking administration

Phenomena

Isolation Level

Implementation aspects

Basics I

What is a transaction?

- Sequence of SQL commands
- Data(base) modifications are an atomic unit

Commit → O.K. accept and fix changes

Rollback → Undo changes

A transaction is a sequence of one or more processing steps. It refers to database objects such as tables, views, joins and so forth.

Here, the following properties must be fulfilled:

- **Indivisibility**

  A transaction is atomic or, in other words, it will either be completely (all of its operations) executed or not at all ("All or nothing principle"). Example: there are no employees without salary.

- **Consistency**

  The defined integrity conditions remain fulfilled. For example, each employee has a personnel number.

- **Isolation**

  The operations within the transaction are isolated from the operations of other transactions.

- **Permanency**

  Changes that transactions have made to objects must be persistent following a system crash, for example.

ACID condition= Atomic, Consistent, Isolation, Durable.

Basics II

Concurrent (parallel) transactions

- Concurrent access to the same database object

➡ **Synchronization logic is required!**

If several transactions want to access the same objects concurrently, these accesses must be synchronized with the help of lock management.

Since the database system allows concurrent transactions to access the same database objects, locks are required to isolate individual transactions.

Locking an object means that other transactions are not able to use it in certain ways.

The more locks that are set, and the longer these stay in place, the less concurrency is possible in database operation.

All locks are released by the end of the transaction at the latest.

Basics III

Locking objects are

- Table rows (ROW)
- Tables (TAB)
- Database catalog (SYS)

Activation

- Implicit
- Explicit

Locking management handles three types of objects:

- Records
- Tables
- Database catalog entries

Requesting locks implicitly

You can choose the lock type by specifying an isolation level when opening the database session. The database system then requests locks implicitly during processing of an SQL statement in accordance with the specified isolation level. All changing SQL statements (such as INSERT, UPDATE, DELETE) always request an exclusive lock.

Requesting locks explicitly

You can use the LOCK statement to explicitly assign locks to a transaction. You can specify a LOCK option in an SQL statement to lock individual rows in a table. This is possible in every isolation level. You can use the LOCK option to temporarily change the isolation level for an SQL statement.

Locking Types

SHARE lock (shared, multiple access)

- Alternate transaction may access the object for reading but not for writing purpose

EXCLUSIVE lock (exclusive access)

- Alternate transactions may access the object for reading pupose but only without a lock (dirty read)

OPTIMISTIC lock

- A transaction (t1) might change an object if and only if no alternate transaction has changed this object after it has been read (by t1, setting the optimistic lock)

Read locks (share locks) refer to a row or a table.

- Once a shared lock is assigned to a transaction for a particular data object, concurrent transactions can access the object but not modify it. Other transactions can set a shared lock, but not an exclusive lock for this object.

Read locks (share locks) refer to a row or a table.

- Once an exclusive lock is assigned a transaction for a particular database object, other transactions cannot modify this object. Transactions that check for the presence of exclusive locks, or that want to set exclusive or shared locks, conflict with the existing exclusive lock of another transaction. You cannot access the locked object.

Optimistic lock on a row level

- An update operation on a row is only actually performed if this row has not been changed in the meantime by a concurrent transaction. If the update operation was successful, an exclusive lock is set for this row. If the update operation was not successful, it should be repeated after reading the row again with or without an optimistic lock. In isolation level 0, an explicit lock must be specified for the new read operation. In this way, it can be ensured that the update is done to the current state and that no modifications made in the meantime are lost.
- It only makes sense to use an optimistic lock if one of the isolation levels 0, 1 or 10, or 15 has been assigned. An optimistic row lock must be explicitly requested by specifying a LOCK statement. A request can conflict with an exclusive lock only.

Locking Types

Can an alternate transaction ... ?	A transaction holds ...					
	EXCL	SHARE	EXCL	SHARE	EXCL	SHARE
	Table lock		Row lock		Catalog lock	
lock this table EXCLUSIVE	NO	NO	NO	NO	NO	YES
lock this table SHARE	NO	YES	NO	YES	NO	YES
lock any row of this table EXCLUSIVE	NO	NO			NO	YES
lock an already locked row EXCLUSIVE			NO	NO		
lock another row EXCLUSIVE			YES	YES		
lock any row of this table SHARE	NO	YES			NO	YES
lock a row SHARE			NO	YES		
lock another row SHARE			YES	YES		
change the table definition in the catalog	NO	NO	NO	NO	NO	NO
read the table definition from the catalog	YES	YES	YES	YES	NO	YES

The above table provides an overview of possible parallel read locks (share locks) and write locks (exclusive locks).

A lock collision exists in the cases which are marked with "No"; i.e., after having requested a lock within a transaction, the user must wait for the lock to be released until one of the above situations or one of the situations that are marked with "Yes" in the matrix occurs.

Additionally, the following applies:

- If no lock has been assigned to a transaction for a data object, then a shared or exclusive lock can be requested within any transaction, and the lock is immediately assigned to the transaction.
- If a shared lock has been assigned to a transaction T for a data object, and if no lock has been assigned to any concurrent transaction for this data object, then the transaction T can request an exclusive lock for this data object and the lock is immediately assigned to this transaction.
- If an exclusive lock has been assigned to a transaction for a data object, then a shared lock can, but need not be requested for this transaction.

Status of a Lock

Normal

The lock is hold until the end of the transaction. It can, as the case may be, be released explicitly.

Consistent

During a table scan a previously received row lock is released if in return another row of the same table gets locked.

Exclusive until end-of-transaction (eot excl)

A lock has been implicitly set during a write order and for consistency reasons has to be kept until the end of transaction (COMMIT or ROLLBACK).

Temporary

In addition to row locks, a table can be locked SHARE for the duration of a mass command (e.g. update).

Configuring Lock Management

DB Kernel Parameters (I)

- MaxSQLLocks — Max. number of locks
- MaxUserTasks — Max. number of concurrent users
- RequestTimeout — Max. waiting time for receiving a lock (in seconds)
- DeadlockDetectionLevel — Depth level for detecting deadlock cycles

- In versions smaller than 7.7 the parameters had the following names: MAXLOCKS, MAXUSERTASKS, REQUEST_TIMEOUT und DEADLOCK_DETECTION

If a lock request collides with an existing lock:

- the user waits on the existing lock, OR
- an error message is returned for the existing lock.

If the user has to wait (default), he will receive an error message after the lock request has timed out.

Timeouts are updated every 30 seconds by the Timer Task.

A deadlock occurs when two or more users mutually prevent each other from proceeding. Deadlocks are recognized down to a certain depth in the database. The users involved in the deadlock receive an error message. The deadlock is resolved.

Deadlocks that were not recognized by the system are resolved by the timeouts (transactions will be rolled back).

Lock Escalation

Transfer row locks to a table lock

- if around 20% of the lock list entries (MaxSQLLocks) are used by one single transaction on one table
- if the number of row locks per transaction exceeds RowLocksPerTransactionThreshold % of MaxSQLLocks

Reacting to collisions during escalation

- Continue by setting further row locks if other concurrent transactions work on the same table.
- Block execution if the mass command requests more locks than available in lock

A mass command is an SQL statement that affects multiple records.

Example: Update PERSONAL set SALARY (GEHALT) = SALARY* 1.5 where GENDER (GESCHLECHT) = "female" (weiblich)

Default value of the parameter RowLocksPerTransactionThreshold is 50. The old name was ROW_LOCKS_PER_TRANSACTION.

System Monitoring (Lock Management)

System tables

- sysdba.lockstatistics
- sysdba.lockliststatistics
- sysdba.transactions

Database console

- x_cons <DBNAME> sh[ow] act[ive]
- the status Vwait shows:
  Task is waiting to get an SQL lock

DB50

- SQL lock overview and waiting status

System Table SYSDBA.Lockstatistics

■ SESSION	internal session id
■ TRANSCOUNT	internal transaction id
■ PROCESS	task id of bound kerneltask
■ USERNAME	name of the user
■ DATE	Start date of database session that holds the lock
■ TIME	Start time of database session that holds the lock
■ TERMID	Terminal-ID des sperrenden Benutzers
■ REQTIMEOUT	seconds to return RequestTimeout
■ LASTWRITE	seconds since last write activity
■ LOCKMODE	lock entry
■ REQMODE	lock request entry
■ APPLPROCESS	process id of the application process (Client)
■ APPLNODE	computer name (client), where the application runs on
■ SCHEMANAME	name of the table schema
■ OWNER	owner of table
■ TABLENAME	name of table
■ TABLEID	table-ID
■ ROWIDLENGTH	length of locked key
■ ROWID	locked key
■ ROWIDHEX	hexadecimal representation of locked key
■ ...	

The system table LOCKSTATISTICS describes the current lock entries and entries for lock requests.

Using the system table LOCKSTATISTICS you can determine the following database information, among other things:

- All locks that are held on a table
- All locks that the current user is holding during his database session (if this is the current user (DBA user) or database system administrator (SYSDBA user), then all locks are displayed).

Users that belong to other user classes only see the locks held by that one user.

System Monitoring (Lock Management)

Views on sysdba.lockstatistics

- DOMAIN.LOCKS and DOMAIN.LOCK_HOLDER
  show all active locks
- DOMAIN.LOCK_REQUESTOR
  shows all lock requests
- DOMAIN.LOCK_WAITS
  shows owners of current lock related to current lock requests

System Table SYSDBA.Lockliststatistics

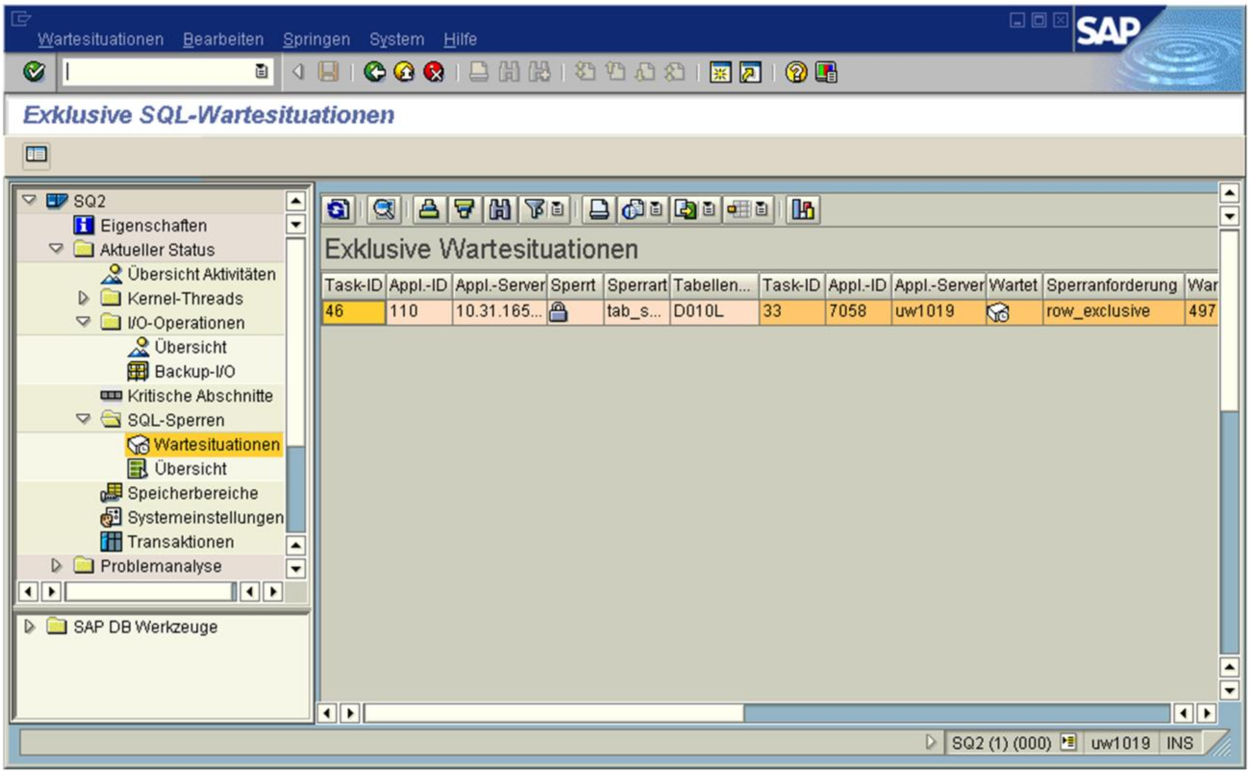
<SYSDBA>.LOCKLISTSTATISTICS

- maximum number of lock entries as defined for locklist
- number of currently used entries
- average number of used entries
- maximum number of used entries
- threshold value for lock escalation
- number of transactions that hold locks
- number of transactions that are requesting locks

You will find a table description of all columns in the system table manual The following lists only particular columns:

- MAX LOCKS contains the number of available locks in the lock list
- `USED ENTRIES` contains the number of entries for locks and lock requests
- `AVG USED ENTRIES` contains the average number of entries used for locks and lock requests
- `MAX USED ENTRIES` contains the maximum number of entries used for locks and lock requests
- `LOCK ESCALATION VALUE` contains the number of table rows from which the lock rows are converted into table locks (lock escalation)
- `SQL LOCK ESCALATIONS` shows the number of escalations occurring so far.
- `LOCK COLLISIONS` shows the number of collisions occurring so far for lock requests.
- `DEADLOCKS` shows the number of deadlocks that have been recognized and resolved by the database system so far.
- `TRANSACTIONS HOLDING LOCKS` contains the number of transactions with assigned locks
- `TRANSACTIONS REQUESTING LOCKS` contains the number of transactions requesting locks

Transaction DB50
Exclusive SQL wait situations



Display of the current wait situations

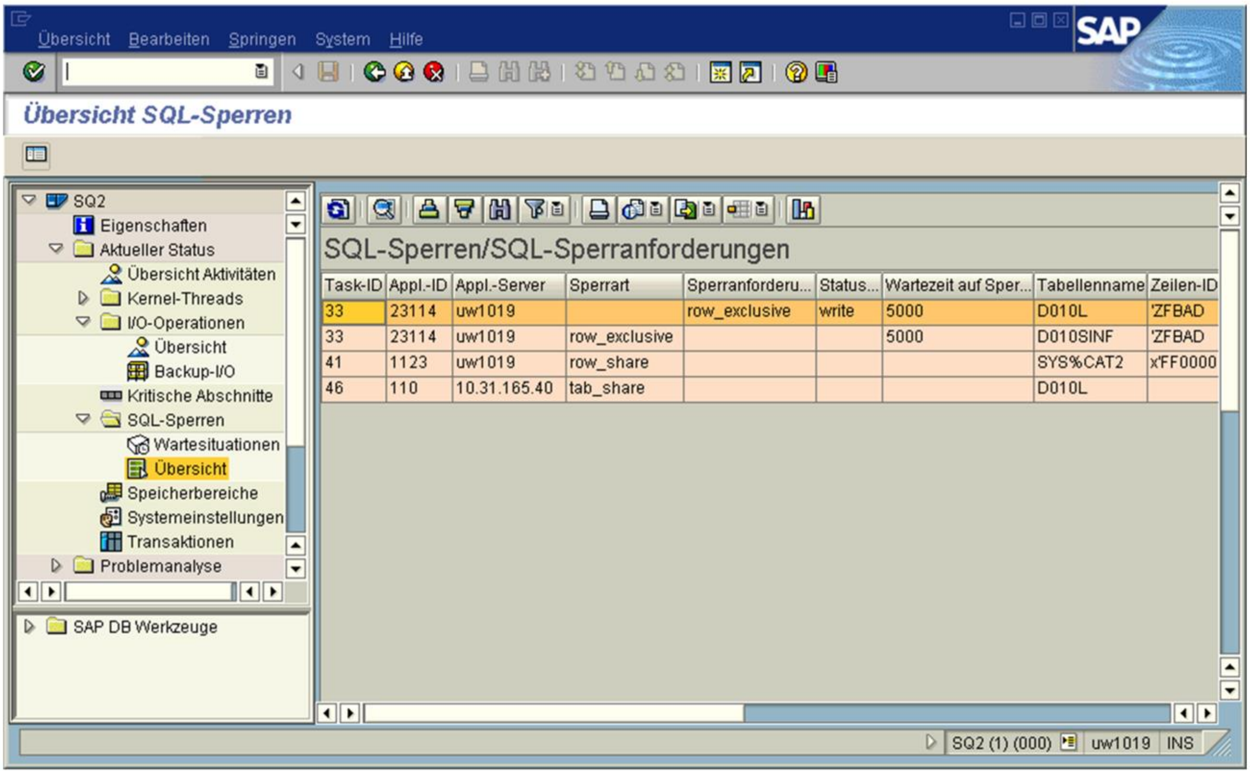
Task 33 waits for a lock, which can then be assigned only once task 46 has provided the shared table lock.

Exclusive locks prevent other users from accessing the locked entry. These locks can significantly interfere with the performance of the SAP system and the database system.

Procedure to determine the user who triggered the lock

- The column "Appl.ID" displays the process ID of the work process on the application server "Appl.Server". You will find the corresponding SAP work process in transaction SM51/SM50 or SM66.
- The corresponding task (here, task 46) can be aborted in the task manager under "Kernel Threads".

General Overview DB50
Overview SQL Locks



Task-ID	Appl.-ID	Appl.-Server	Sperrart	Sperranforderu...	Status...	Wartezeit auf Sper...	Tabellenname	Zeilen-ID
33	23114	uw1019		row_exclusive	write	5000	D010L	'ZFBAD
33	23114	uw1019	row_exclusive			5000	D010SINF	'ZFBAD
41	1123	uw1019	row_share				SYS%CAT2	x'FF0000
46	110	10.31.165.40	tab_share				D010L	

Display of all active and requested database locks.

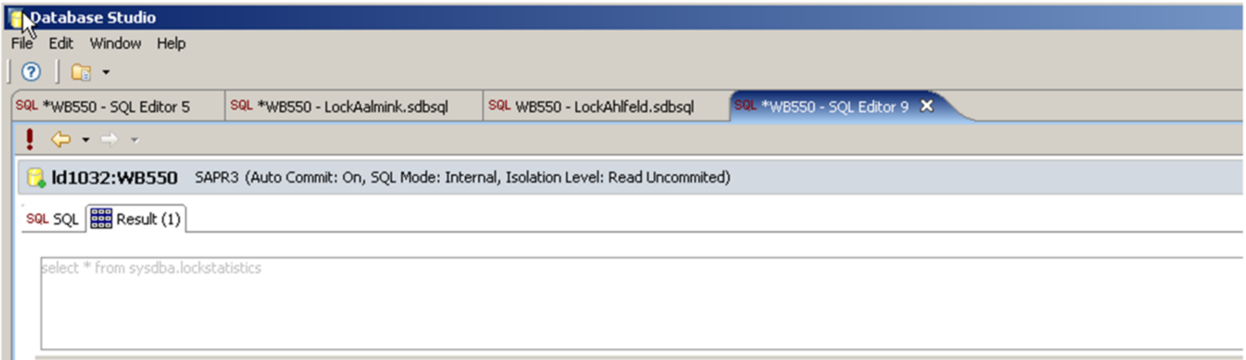
Exclusive locks prevent other users from accessing the locked entry. These locks can significantly interfere with the performance of the SAP system and the database system.

The system displays detailed information about the locks currently set. This display can be very long in a running SAP system. Therefore, always display the analysis of SQL locks from the overview of wait situations (exclusive).

Task T33 requests a write lock on a record belonging to the table D010L.

Task T46 holds a table lock on table D010L.

SYSDBA.Lockstatistics in Database Studio



	SESSION	TRANSCOUNT	SUB_TRANS	WRITE_TRANS	PROCESS	USERNAME	DATE	TIME	TERMID	REQTIMEOUT	LASTWRITE	LOCKMODE
1	162	572	0	?	220	SAPR3	2007-07-30	15:17:18	java@1589559	?	?	row_share
2	162	572	0	?	220	SAPR3	2007-07-30	15:17:18	java@1589559	?	?	row_excl...
3	162	572	0	?	220	SAPR3	2007-07-30	15:17:18	java@1589559	?	?	row_excl...
4	159	583	0	?	221	SAPR3	2007-07-30	15:16:22	java@d81c91	4915	?	?

• • •

LOCKMODE	LOCKSTATE	REQMODE	REQSTATE	APPLPROCESS	APPLNODE	SCHEMAN...	OWNER	TABLENAME	TABLEID	ROWIDLEN...	ROWIDHEX	ROWI
row_share	?	?	?	0	10.18.107.18	?	?	?	0000000000000000	9	00FFFF00000000001C000000000...	?
row_excl...	?	?	?	0	10.18.107.18	SAPR3	SAPR3	ZZTELE	00000000000002...	69	2041616C6D696E6B20202020202...	'Aalm.
row_excl...	?	?	?	0	10.18.107.18	SAPR3	SAPR3	ZZTELE	00000000000002...	69	2041686C66656C6420202020202...	'Ahlf..
?	?	row_ex...	?	0	10.18.107.18	SAPR3	SAPR3	ZZTELE	00000000000002...	69	2041686C66656C6420202020202...	'Ahlf..

Special Scenarios

Phenomena

- Dirty Read
- Non Repeatable Read
- Phantom

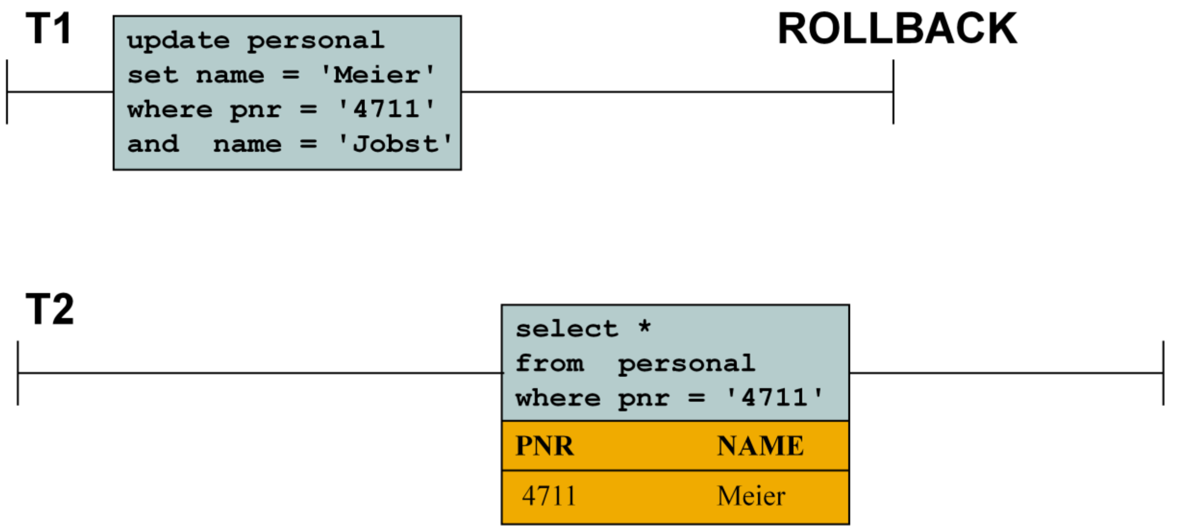
The isolation level plays an important role in the lock activities of the database system. You use the isolation level to specify whether locks are requested or released implicitly, and how.

Your choice of isolation level affects the degree of parallelism of concurrent transactions and the consistency of the data: the lower the value of the isolation level, the higher the degree of parallelism, and the lower the degree of guaranteed consistency

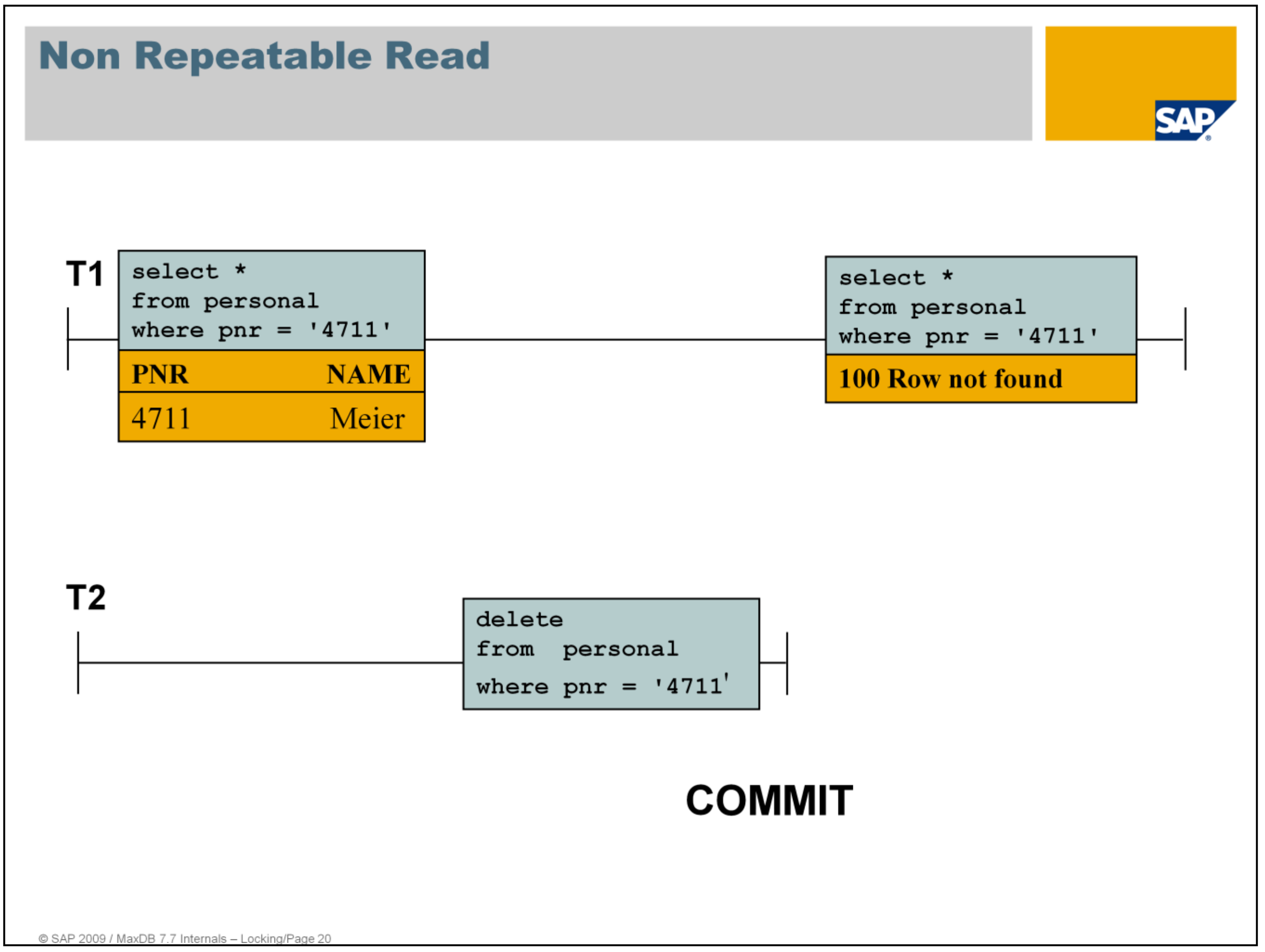
If transactions are competing for access to the same data, then different isolation levels can cause different sorts of inconsistencies. You can find a compromise between parallelism and consistency, while taking into account the requirements of your database application.

When concurrent transactions are processed, inconsistent situations can occur. Try and avoid these situations by configuring the lock behavior and isolation level of the database system accordingly.

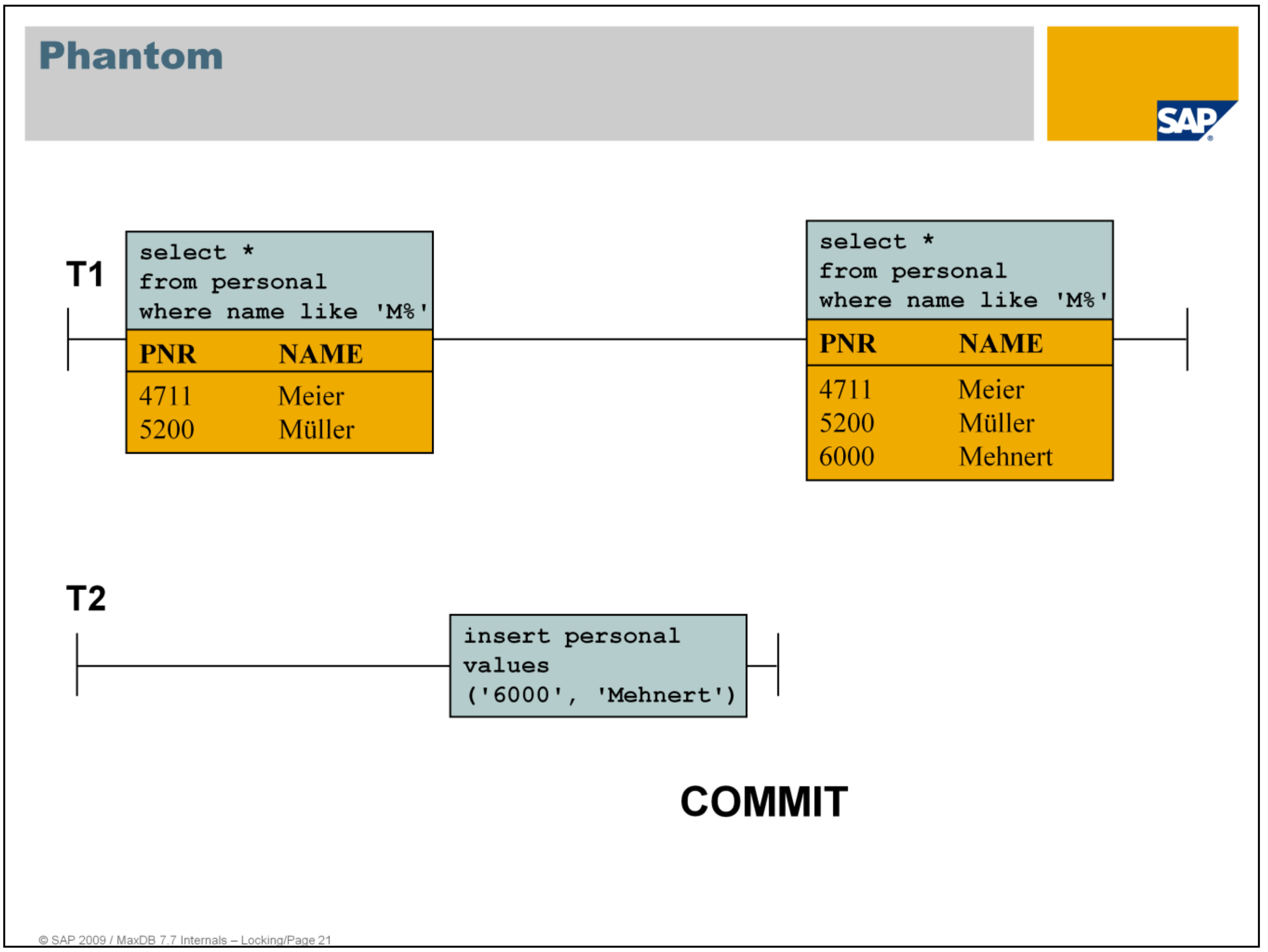
Dirty Read



A row is modified in the course of a transaction T1, and a transaction T2 reads this row before T1 has been concluded with the COMMIT statement. T1 then executes the ROLLBACK statements. In this case, T2 read a row that never actually existed.



Transaction T1 reads a row. Transaction T2 then modifies or deletes this row, and completes the action with the commit statement. If T1 then reads the row again, it either gets the modified row or a message indicating that the row no longer exists.



Transaction T1 executes an SQL statement S that reads a set of rows (M) fulfilling a search condition. Transaction T2 then inserts or modifies data, and produces another row that fulfills this search condition. If T1 then executes the statement S again, the set of rows that is read differs from the set M.

ISOLATION LEVEL 0

Read access

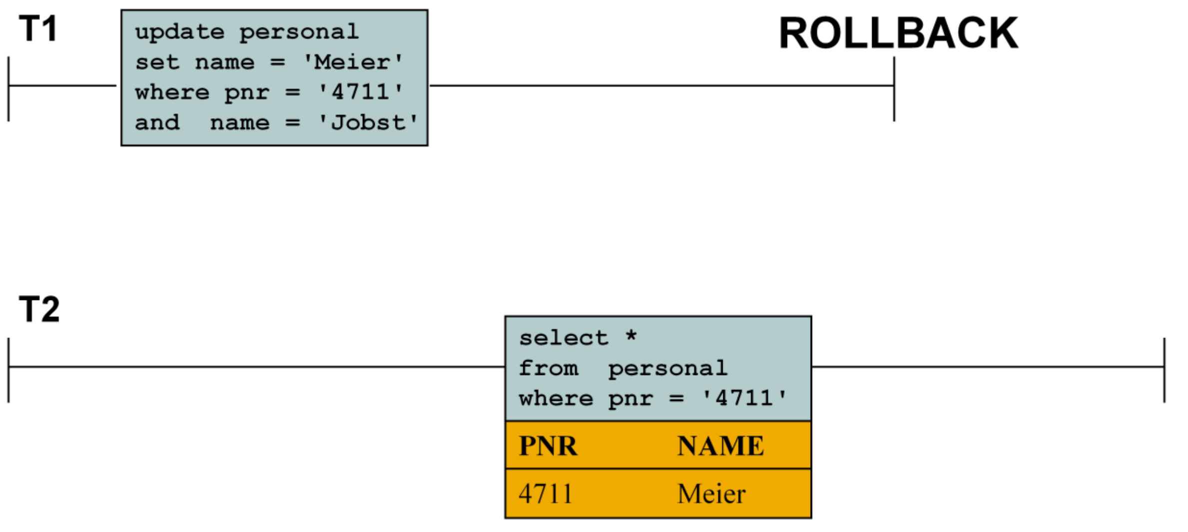
- Rows are read without checking for lock collisions
- It is not guaranteed that
  - a repeated read within the same transaction returns the same result
  - rows once read ever will be committed (become persistent)

Isolation level 0 does not offer any protection against access anomalies.

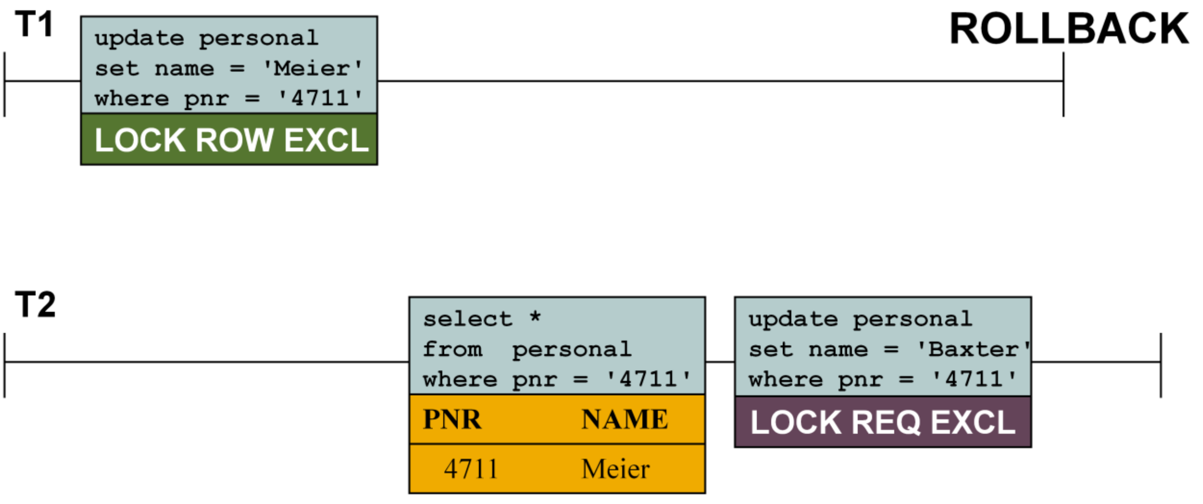
If you specify the isolation level 0 (uncommitted), then rows are read without shared locks being requested implicitly. If a row is then read twice within a transaction, this isolation level does not guarantee that the row has the same state the second time as the first, since it could have been changed by a competing transaction between the two reads.

Furthermore, there is no guarantee that the state of a row that was read has already been recorded in the database using a COMMIT WORK statement.

ISOLATION LEVEL 0



ISOLATION LEVEL 0



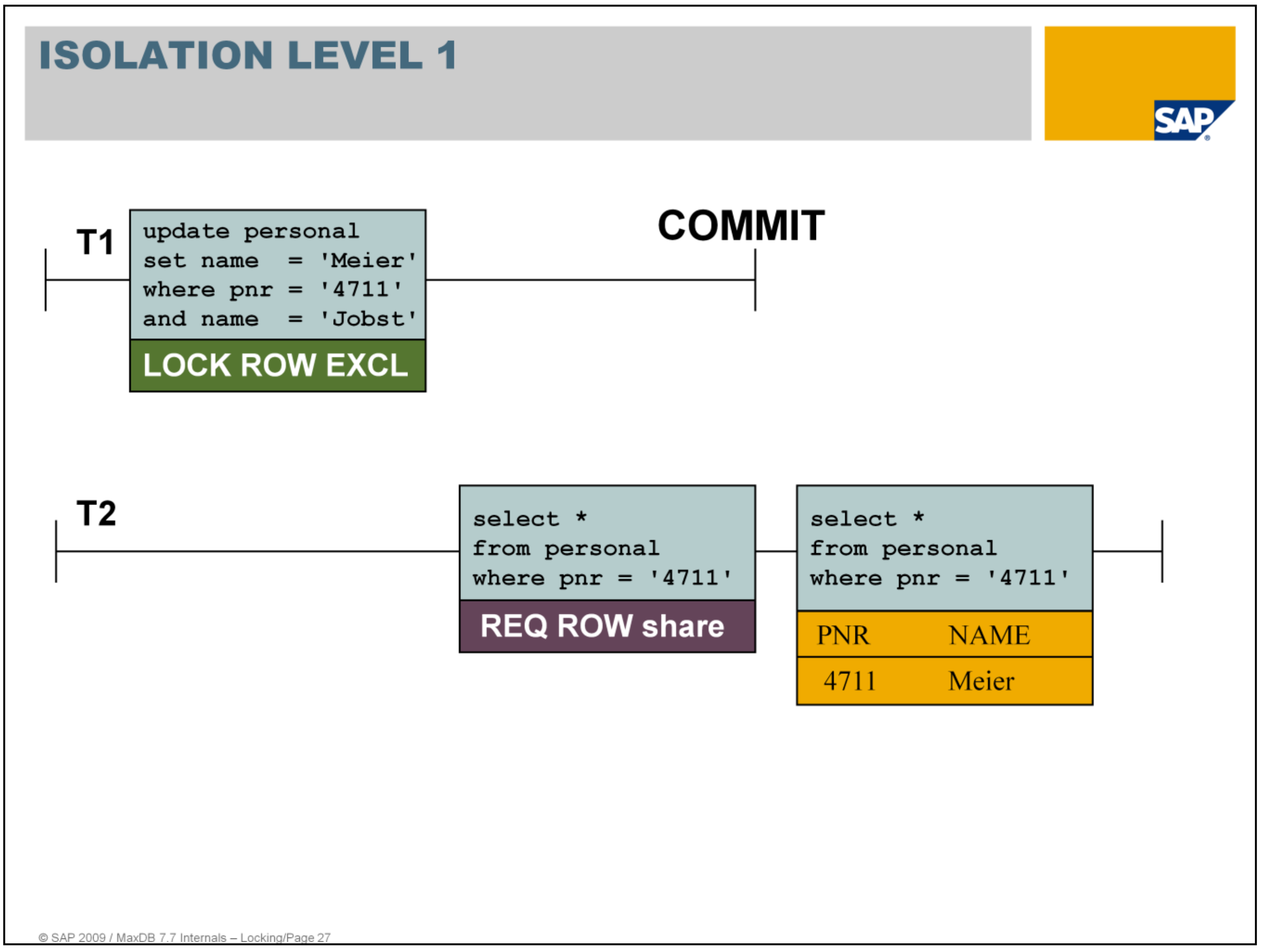
ISOLATION LEVEL 1

Read Access

- Read persistent (committed) rows. Check for collision happens before reading
- In case of a collision a lock request is set. (req row share)
- The lock dispatcher implicitly changes the request into a lock, if the colliding lock is released. The dispatcher uses a priority list to find the optimum user process.

When you retrieve data using an SQL statement, the database system ensures that, at the time each row is read, no exclusive lock has been assigned to other transactions for the given row. However, it is impossible to predict whether an SQL statement causes a shared lock for a row of the specified table and for which row this may occur. In SAP DB versions < 7.4, the share locks were held until the end of the transaction. In version 7.4 and above, the share lock is removed after the record has been read.

Locking of data entities and optimal multi-user operation are in direct conflict with one another. It is not recognizable whether the waiting user is waiting for a lock or whether the system is running poorly.


ISOLATION LEVEL 1
SAP
T1
update personal
set name = 'Meier'
where pnr = '4711'
and name = 'Jobst'
LOCK ROW EXCL
COMMIT
T2
select *
from personal
where pnr = '4711'
REQ ROW share
select *
from personal
where pnr = '4711'
PNR NAME
4711 Meier
© SAP 2009 / MaxDB 7.7 Internals – Locking/Page 27

ISOLATION LEVEL 1

SAP

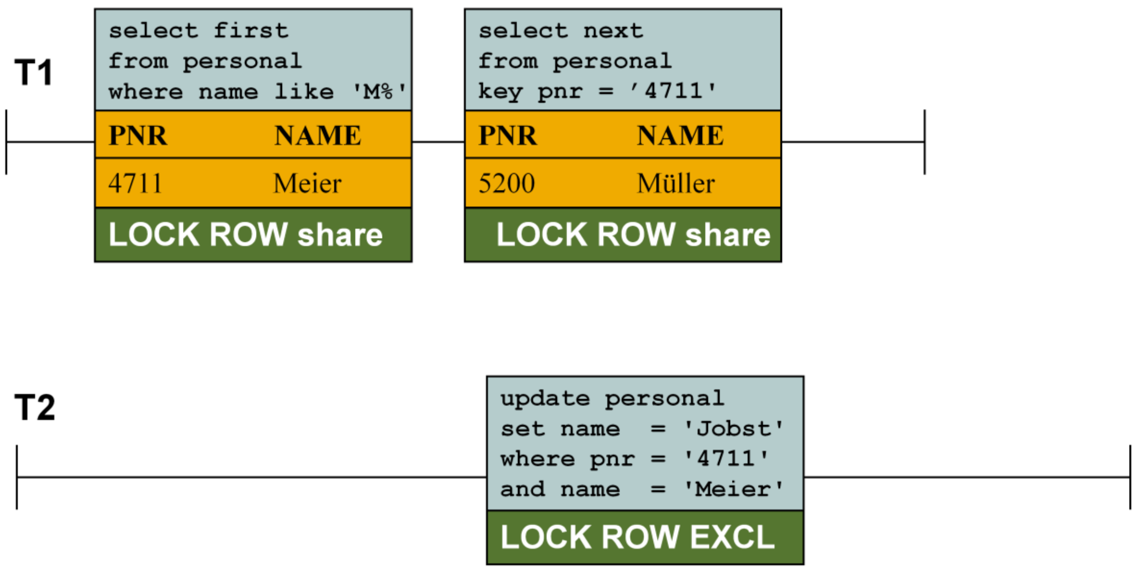
Read Access

- In case of direct key accesses (select direct, select next, select ... key) there will be "moving" share locks

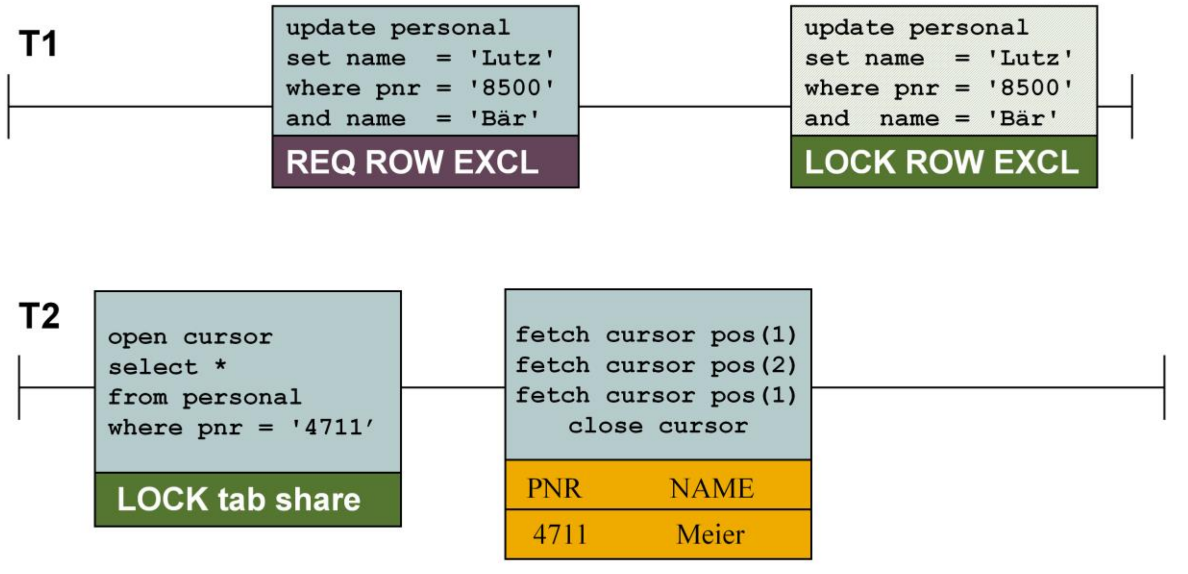
  Reading a row will cause setting a share lock for this row simultaneously with releasing the lock from the last row access

If you specify the isolation level 1 or 10 *(committed),* then a shared lock is assigned to the transaction for a read row Z1 of a table. When in the same table the row Z2 is read, the lock on Z1 is released and a shared lock is assigned to the transaction for the row Z2.

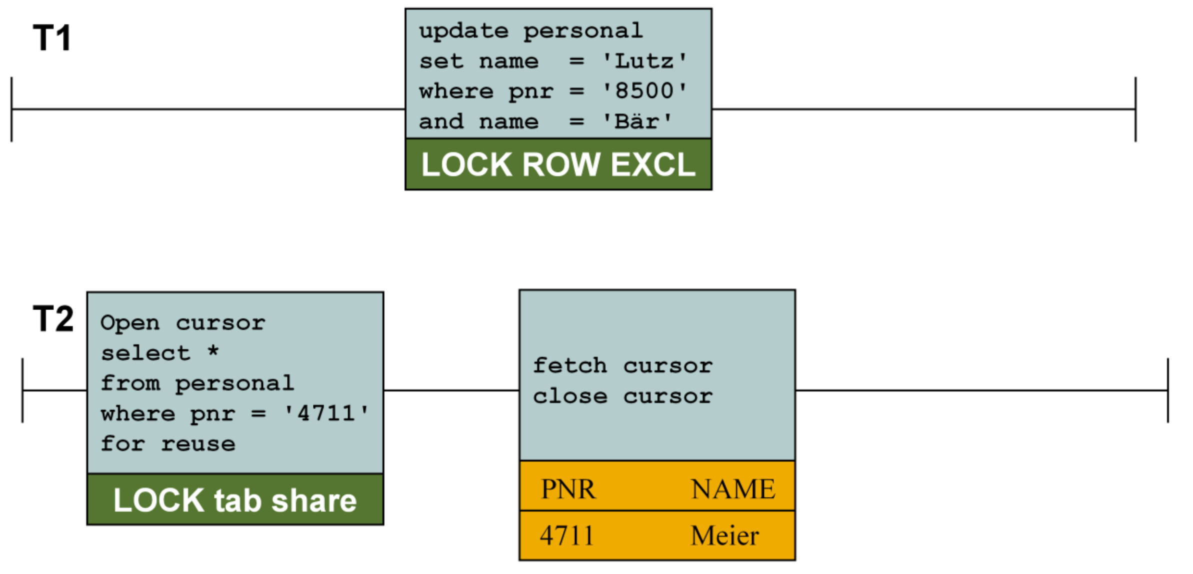
ISOLATION LEVEL 1



ISOLATION LEVEL 15



ISOLATION LEVEL 15



ISOLATION LEVEL 15

Write access

- Goal:
  - modify tables that are committed for the duration of an SQL command
- Implementation:
  - temporary table locks during execution of the SQL command (tab share)
  - exclusive row locks (row excl) on new/updated rows until end of transaction

When inserting, changing or deleting rows, the exclusive locks are assigned implicitly to the transaction for the relevant rows that are not released until the end of the transaction.

ISOLATION LEVEL 2

Read access

- Goal:
  - read from tables that are committed for the duration of an SQL command
  - avoid concurrent follow up modifications to the rows read
- Implementation:
  - temporary table locks during execution of the SQL command (tab share)
  - the rows read are secured from concurrent modifications by using share locks (repeatable read)

Isolation Level 2 safeguards against the "Non Repeatable Read" phenomenon,.

A record that is read multiple times within a transaction always contains the same values.

ISOLATION LEVEL 2

Read access

- the temporary table lock (tab share) will be released
  - after execution of the SQL command, if a temporary result table is created (result is copied)
  - after closing of the result table (close)
  - at the end of the transaction
- the row locks will be released (row share)
  - at the end of the transaction
  - explicitly with an UNLOCK command

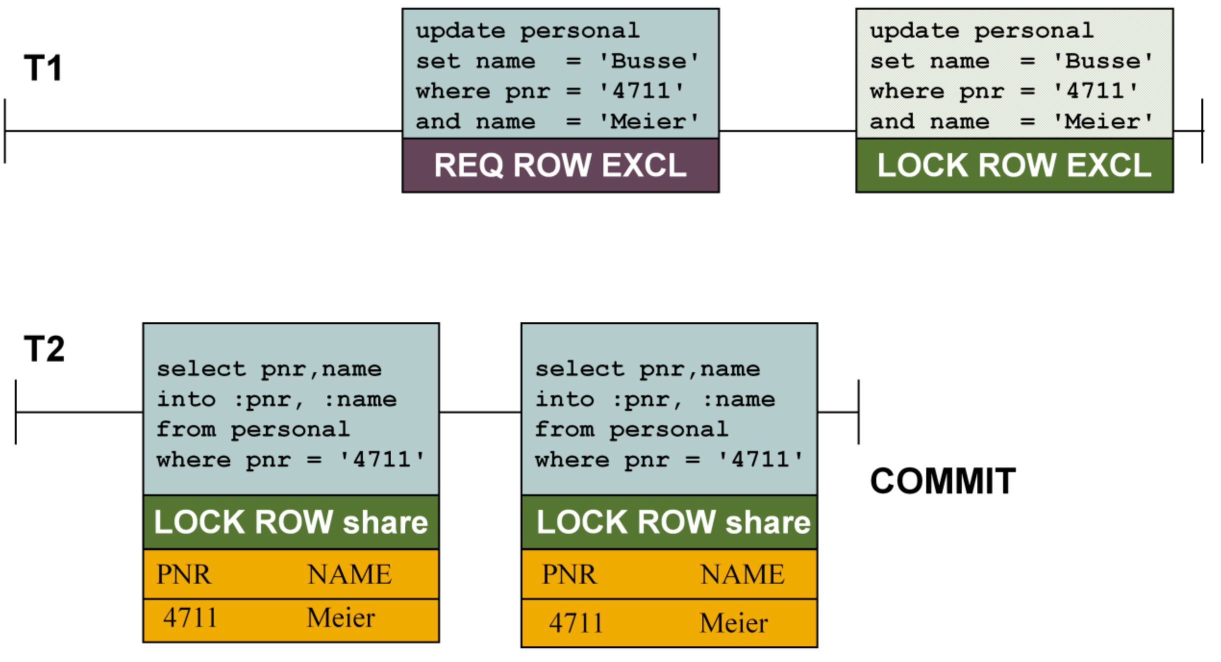
If you specify the isolation level 2 or 20 (repeatable), then shared locks are requested implicitly for all the tables addressed by an SQL statement data query before processing starts.

If an SQL statement generates a result table, which is not physically saved, then these locks are not released until the end of the transaction or when the result table is closed. Otherwise, the locks are released immediately after the SQL statement is processed.

The table shared lock is not assigned to the transaction with SQL statements, where exactly one row in a table is processed that is determined by key specifications or using `CURRENT OF <result_table_name>`.

In addition, an implicit shared lock is assigned to the transaction for each row read while an SQL statement is being processed. These locks can only be released using an UNLOCK statement or by ending the transaction.

ISOLATION LEVEL 2



ISOLATION LEVEL 2

Write Access

- Goal:
  - modify tables that are committed for the duration of an SQL command
- Implementation:
  - temporary table locks during execution of the SQL command (tab share)
  - exclusive row locks (row excl) on new/updated rows until end of transaction

When inserting, changing or deleting rows, the exclusive locks are assigned implicitly to the transaction for the relevant rows that are not released until the end of the transaction. No locks are set for the whole table, however.

ISOLATION LEVEL 3

Read access

- Goal:
  - read from tables that are committed for the duration of an SQL command
  - avoid concurrent follow up modifications to the rows read during current transaction (phantom)
- Implementation:
  - table locks (tab share)
  - release of share locks
    - at the end of the transaction
    - explicitly with an UNLOCK command

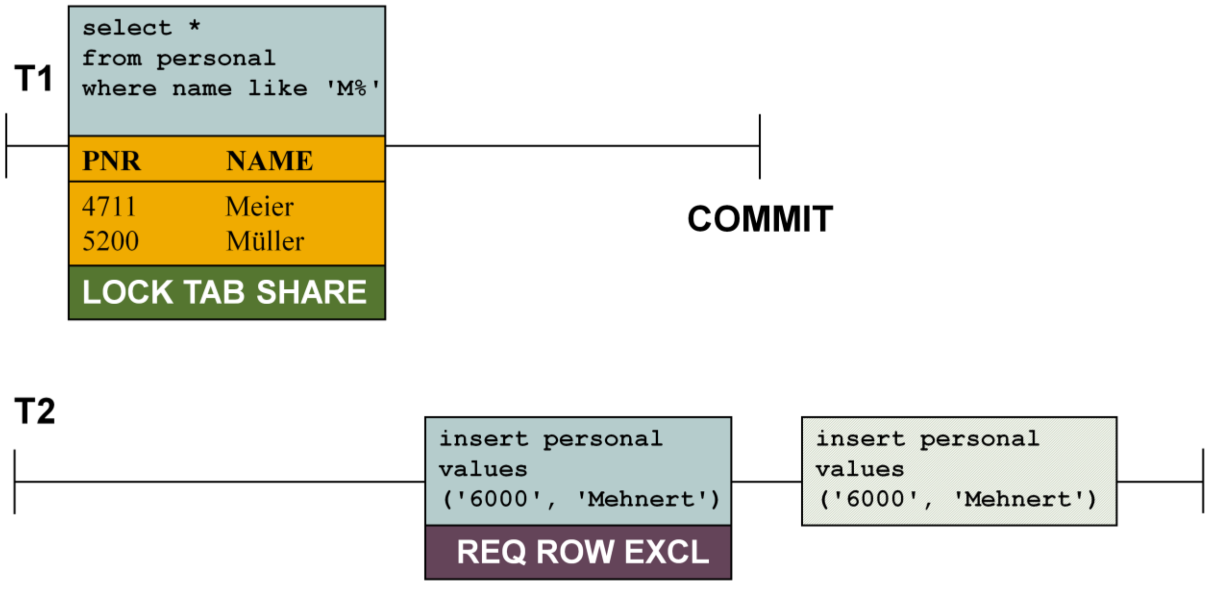
If you specify the isolation level 3 or 30 (serializable), then a table shared lock is implicitly assigned to the transaction for every table addressed by an SQL statement.

These shared locks can only be released by ending the transaction. This table shared lock is not assigned to the transaction with SQL statements, where exactly one row in a table is processed that is determined by key specifications or using CURRENT OF <result_table_name>.

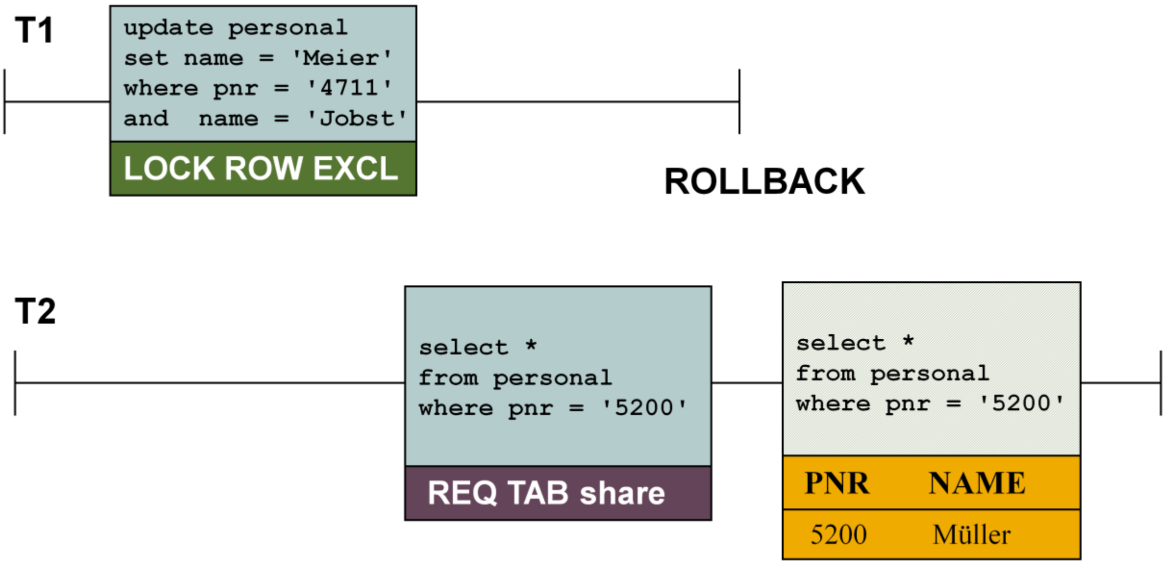
Isolation level 3 safeguards against three types of access anomalies:

- Dirty Read
- Non Repeatable Read
- Phantom

ISOLATION LEVEL 3



ISOLATION LEVEL 3



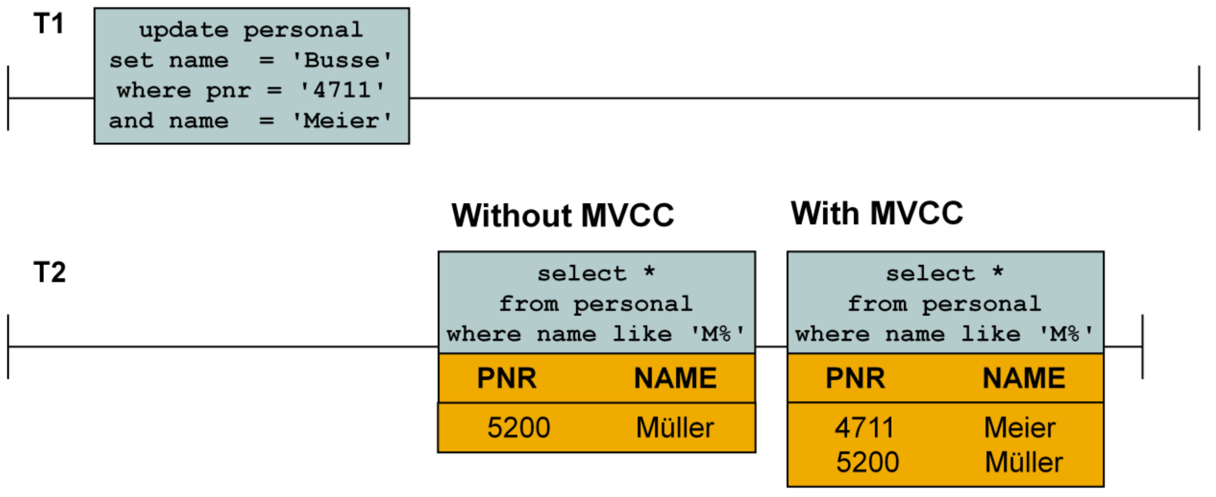
ISOLATION LEVEL 3

Write Access

- Goal:
  - modify tables that are committed for the duration of an SQL command
- Implementation:
  - table locks during execution of the SQL command (tab share)
  - exclusive row locks (row excl) on new/updated rows until end of transaction

When inserting, changing or deleting rows, the exclusive locks are assigned implicitly to the transaction for the relevant rows that are not released until the end of the transaction.

MVCC: Multiversion Concurrency Control
Difference to other DBMs



Supporting MVCC (Mutliversion Concurrency Control) has been implemented for the most part in version 7.7 and is now in testing phase.

Lock Key

Implementation

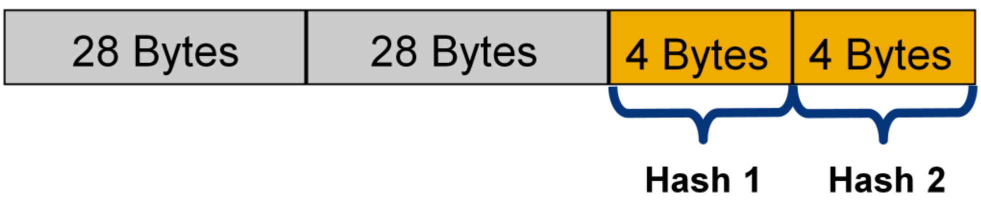
- Locking a row (I)
  - a row is locked by blocking the access via primary key
  - if a unique index exists, this secondary key is locked too (secure the one-to-one relation)

Lock Key Composition

Lock keys have a maximum length of 64 Byte

A lock key consists of 4 parts:

- Start of the primary key (28 Byte)
- Tail of the primary key (28 Byte)
- Generic hash values (2 x 4 Byte)



The unique identifier of a record is the primary key. Locks using long primary key values with variable lengths would be inefficient.

Thus, for each primary key a lock key is generated if the primary key is longer than 64 bytes.

As of version 7.4, the lock key has a maximum length of 64 bytes, which is independent of the 64-bit or 32-bit architecture.

The first 56 bytes of the lock key are created from the combination of the first 28 bytes and last 28 bytes of the primary key.

The last 8 bytes of the lock key are generated using two different hash algorithms and with the help of the entire primary key. The result of the first hash algorithm is stored in the second-to-last 4 bytes and the result of the second hash algorithm in the last 4 bytes. Thus, optimum dispersion of the of the lock key values is ensured.

Locking LOB Columns

Implementation

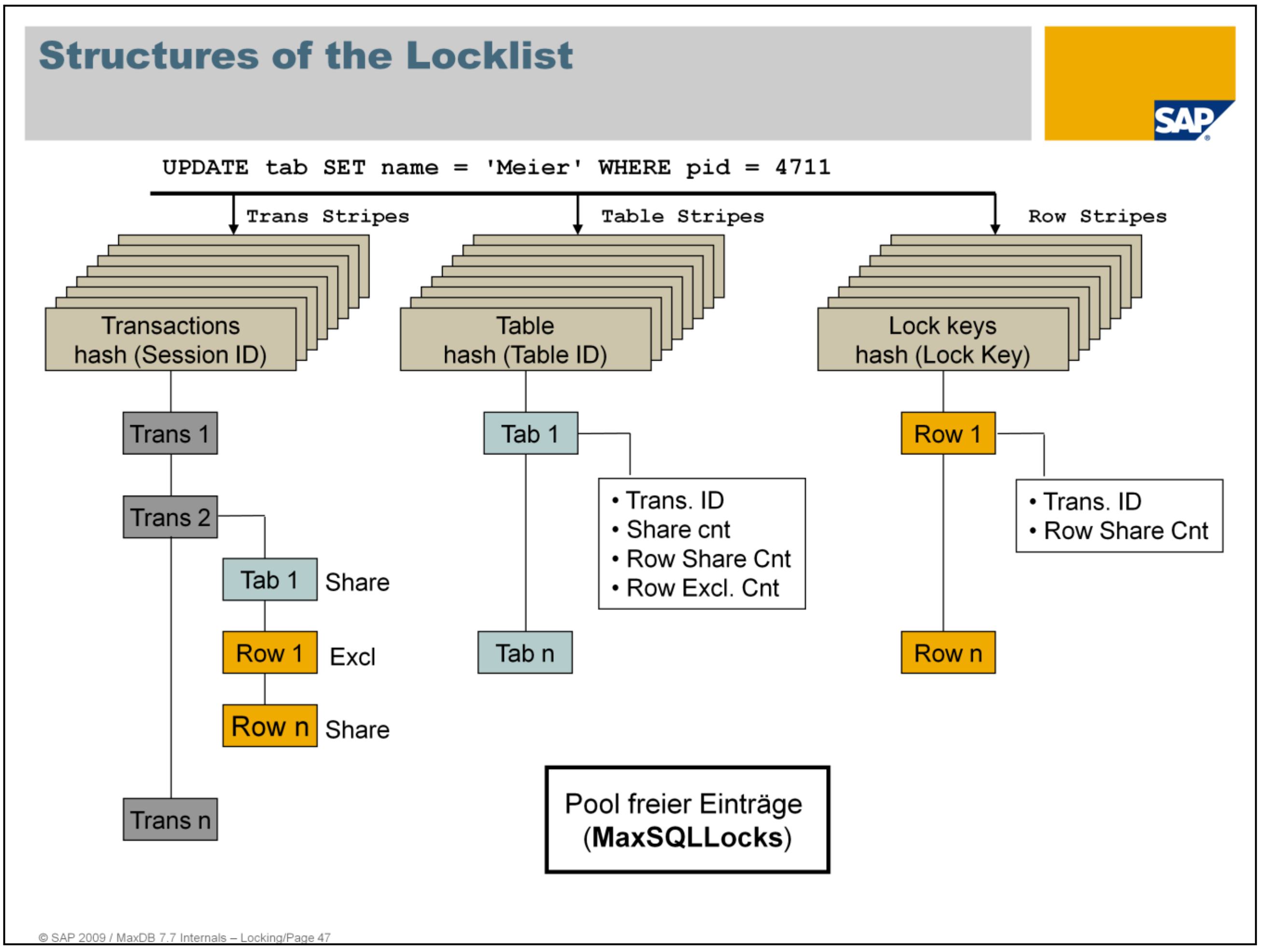
- Locking a row (II)
  - Accesses to LOB (former LONG) values always are secured by locks as reading and writing LOB values can require multiple I/O requests. Thus locking LOBs is independent of the ISOLATION LEVEL)
  - Locks for LOB values are realized as table locks (tab).
  - Each LOB value is identified by a unique TAB-ID, which is used to build the lock entry (i.e. the lock key)
- Read access
  - The database releases the lock immediately after the read access.

Locklist

Concept (I)

- Structure:
  - segmental list of transactions
  - segmental list of tables
  - segmental list of lock keys

- Views into the locklist
  - local view on transactions
  - global segment oriented view on tables
  - global segment oriented view on lock keys



Procedure for generating a lock entry:

- Check whether the private transaction view already contains the lock entry (unprotected sequential search).
- If the lock entry is not contained in the private transaction view, the segment assigned to the table of the global table view is checked with respect to a lock collision (hash access).
- In the case of table locks, set the lock or make a lock request
- If the lock entry is not contained in the private transaction view, the segment that is assigned to the table of the global table view is checked with respect to a lock collision (hash access).
- Set the lock or make a lock request for row locks.

The transaction ID displays the exclusive lock in the table and lock key view.

In the table view, Row Share and Exclusive Counter are used to calculate the escalation.

The number of segments of each view is set using the parameters TransactionLockManagementStripes (_TRANS_RGNS), TableLockManagementStripes (_TAB_RGNS)und RowLockManagementStripes (_ROW_RGNS). There are 8 segments, by default.

Background: Locklists

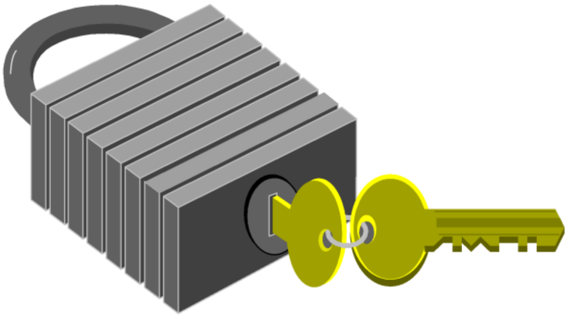
Advantages

- concurrent operation supported by multi-layered segmentation
- hash accesses reduce lookup-time for lock keys

Disadvantages

- memory consumption
- single user operation requires more segment accesses

End



THE BEST-RUN BUSINESSES RUN SAP™