SAP® MaxDB™
Analysis of SQL Locking Situations

Expert Session 12

MaxDB/liveCache Development Support
Christiane Hienger
Bettina Laidler
September 7, 2010

THE BEST-RUN BUSINESSES RUN SAP™

Agenda

1. **Introduction**
   - 1.1. What is a transaction?
   - 1.2. ACID model
   - 1.3. Why must SQL locks be set on database objects?
2. **Lock Objects And Locking Types**
   - 2.1. Which lock objects do exist in SAP MaxDB?
   - 2.2. Locking types
3. **Phenomena And Isolation Level**
   - 3.1. Dirty read
   - 3.2. Non repeatable read
   - 3.3. Phantom
   - 3.4. Isolation level 0
   - 3.5. Isolation level 1
   - 3.6. Isolation level 15
   - 3.7. Isolation level 2
   - 3.8. Isolation level 3

© 2010 SAP AG. All rights reserved. / Page 2

The target group of this session are customers with SAP Applications running on SAP MaxDB, SAP partners and SAP employees who support SAP MaxDB customers .

This session is based on MaxDB version 7.8 database called EXPERTDB which is integrated into the internal development System S70.

This session consists of an theoretical part and a second practical part where analysis of lock situation are shown.

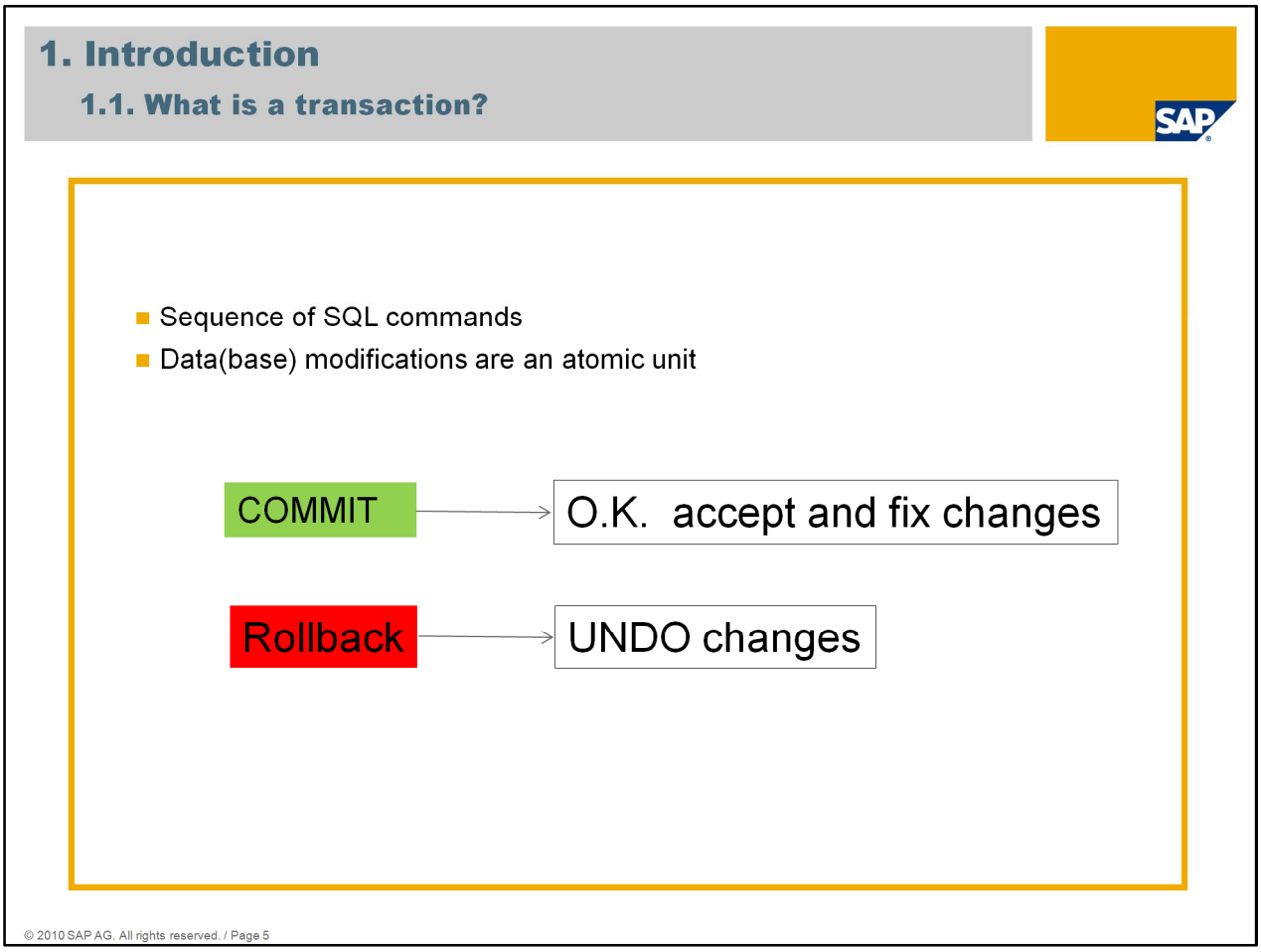
Agenda

4. **System Monitoring**
   - 4.1. Monitoring lock situations
   - 4.2. System tables and views
5. **Lock Escalation**
   - 5.1. What is a lock escalation?
   - 5.2. DBACockpit Activities Overview
   - 5.3. Parameter MaxSQLLocks
6. **Deadlock**
   - 6.1. What is a deadlock?
   - 6.2. Demo with DeadlockDetectionLevel = 4
   - 6.3. Demo with DeadlockDetectionLevel = 0
7. **Which Application Blocks The System?**
   - 7.1. Demo
8. **Summary**
   - 8.1. Configuring lock management
   - 8.2. Error messages
   - 8.3. Information resources

© 2010 SAP AG. All rights reserved. / Page 3

Agenda

All rights reserved. / Page 4

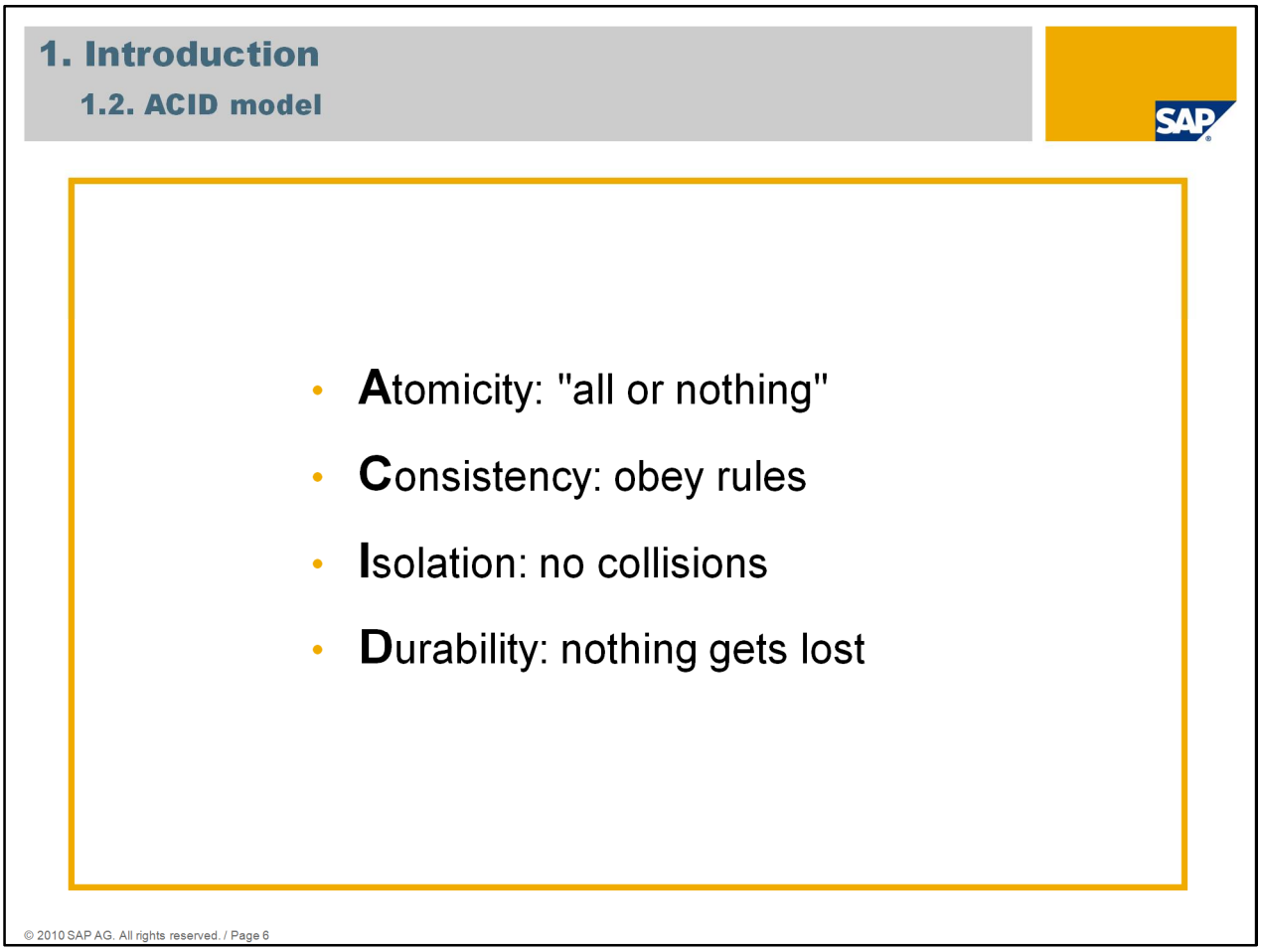


A transaction is a sequence of one or more processing steps (SQL commands). It refers to database objects such as tables, views, joins, and so on. The database modifications must follow an “all or nothing” rule. If one part of the transaction fails, the entire transaction fails.

When all SQL commands of a transaction are executed successfully the transaction can be closed with COMMIT work.

If errors happen during the processing of a transaction the entire transaction will be rolled back.

The next slides show the requirements which must be fulfilled (ACID condition = atomic, consistent, isolation, durable).



The ACID model is one of the oldest and most important concepts of database theory. It sets forward four goals: atomicity, consistency, isolation and durability. No database that fails to meet any of these four goals can be considered reliable.

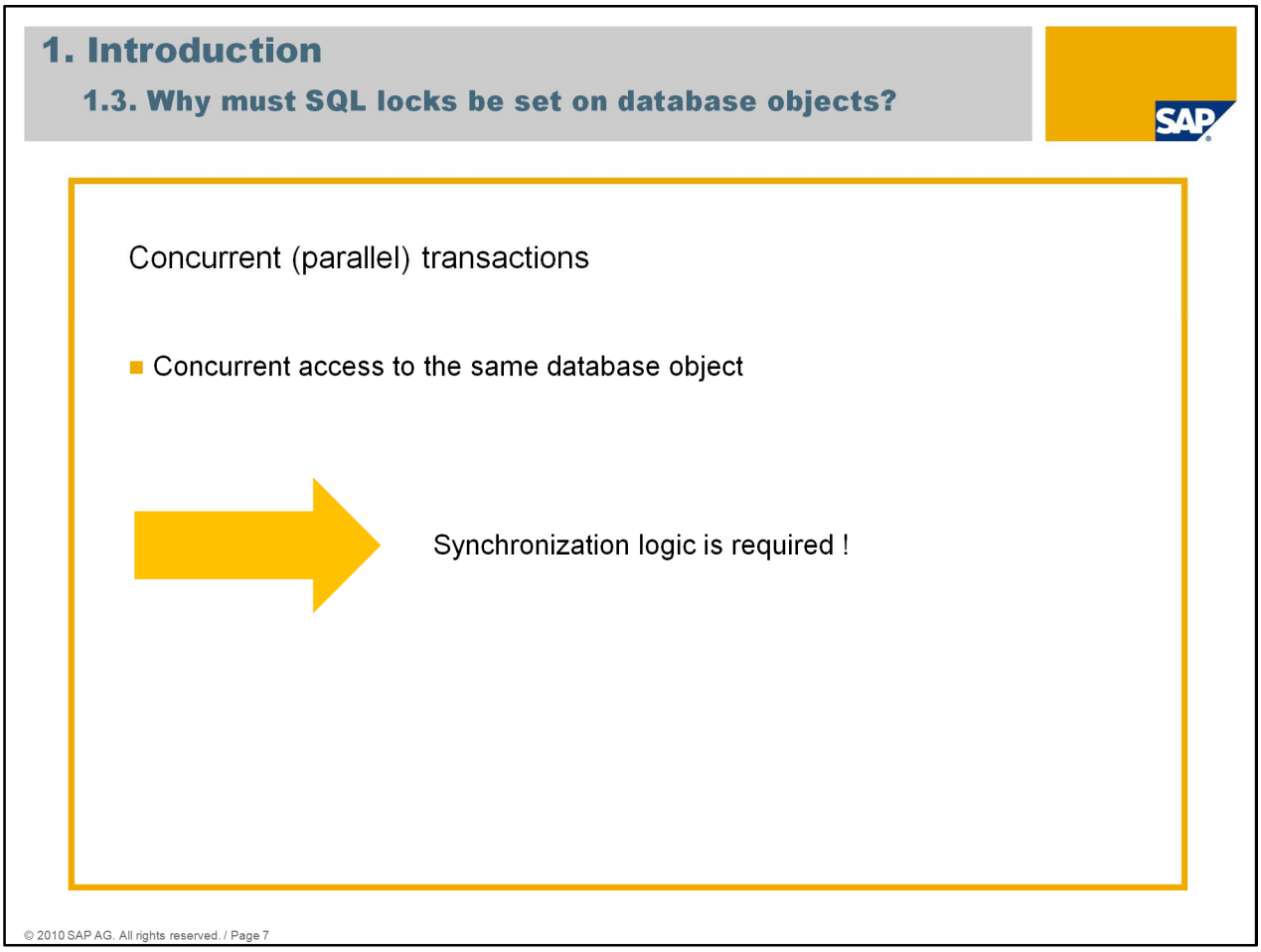
Indivisibility (atomic):
A transaction is atomic, which means that it is either executed completely (all its operations) or not at all (all or nothing principle).

For example: There is no employee without a salary.

Maintaining **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.

Durability:
Changes that completed transactions made to objects must be permanent (persistent), for example, even if a system crashes.



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.

To lock an object means to lock this object from certain forms of use by other transactions.

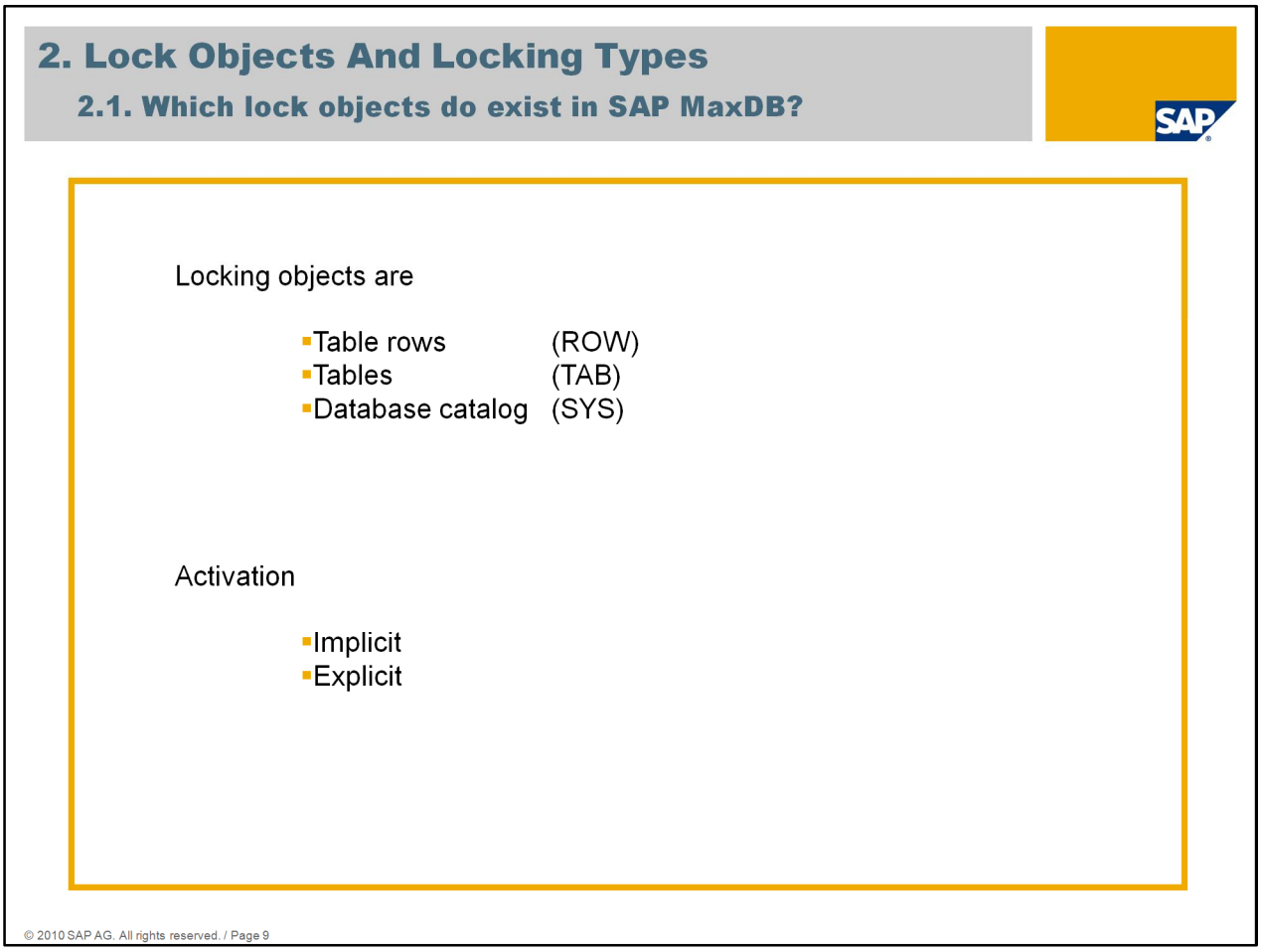
The more locks that are set, and the longer these stay in place, the less concurrency is possible in database operation.

All locks are released by the end of the transaction at the latest. (COMMIT, ROLLBACK, COMMIT WORK RELEASE, ROLLBACK WORK RELEASE).

Agenda

1. Introduction
2. Lock Objects And Locking Types
3. Phenomena And Isolation Level
4. System Monitoring
5. Lock Escalation
6. Deadlock
7. Which Application Blocks The System?
8. Summary

All rights reserved.



Locking management handles three types of objects:

- Records (ROW) - Individual lines of a table are locked
- Tables (TAB) - The entire table is locked.
- Database catalog entries (SYS) - Database catalog entries are locked.

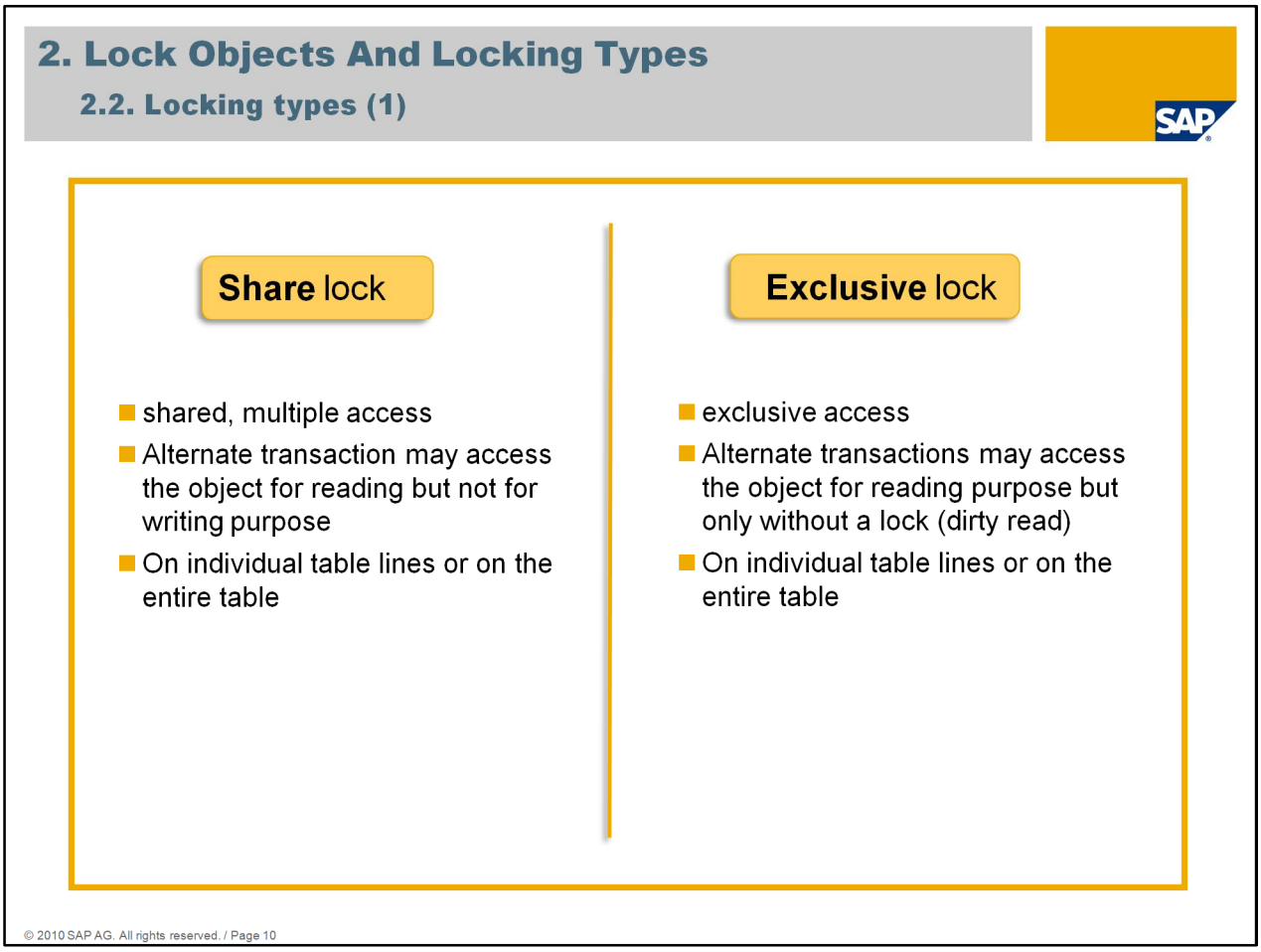
Locks can be requested implicitly by the database system or explicitly by you (using the relevant SQL statements).

Requesting locks implicitly:

All modifying SQL statements (for example INSERT, UPDATE, DELETE) always request an exclusive lock. You can select the lock operation mode by specifying an **isolation level** when you open the database session. Depending on the specified isolation level, locks are then implicitly requested by the database system when the SQL statements are processed.

Requesting locks explicitly:

You can assign locks explicitly to a transaction using the LOCK statement, and you can lock individual table lines by specifying a LOCK option in an SQL statement. This procedure is possible at each **isolation level**. In addition, you can use the LOCK option to temporarily change the isolation level of an SQL statement.



What is a shared lock (Read Locks)?

Shared locks allow other transactions to perform read accesses but not to perform write accesses to the locked object. Other transactions can set further shared locks on this object, but no exclusive locks.

Shared locks can be set on individual table lines or on the entire table.

What is an exclusive lock (Write Locks)?

If an exclusive lock on a database object is assigned to a transaction, other transactions cannot access this object.

Transactions that want to check whether exclusive locks exist or to set exclusive or shared locks collide with the existing exclusive

lock of the other transaction. They cannot access the locked object.

Exclusive locks can be set on individual table lines or on the entire table.

2. Lock Objects And Locking Types

2.2. Locking types (2)

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

© 2010 SAP AG. All rights reserved. / Page 11

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.

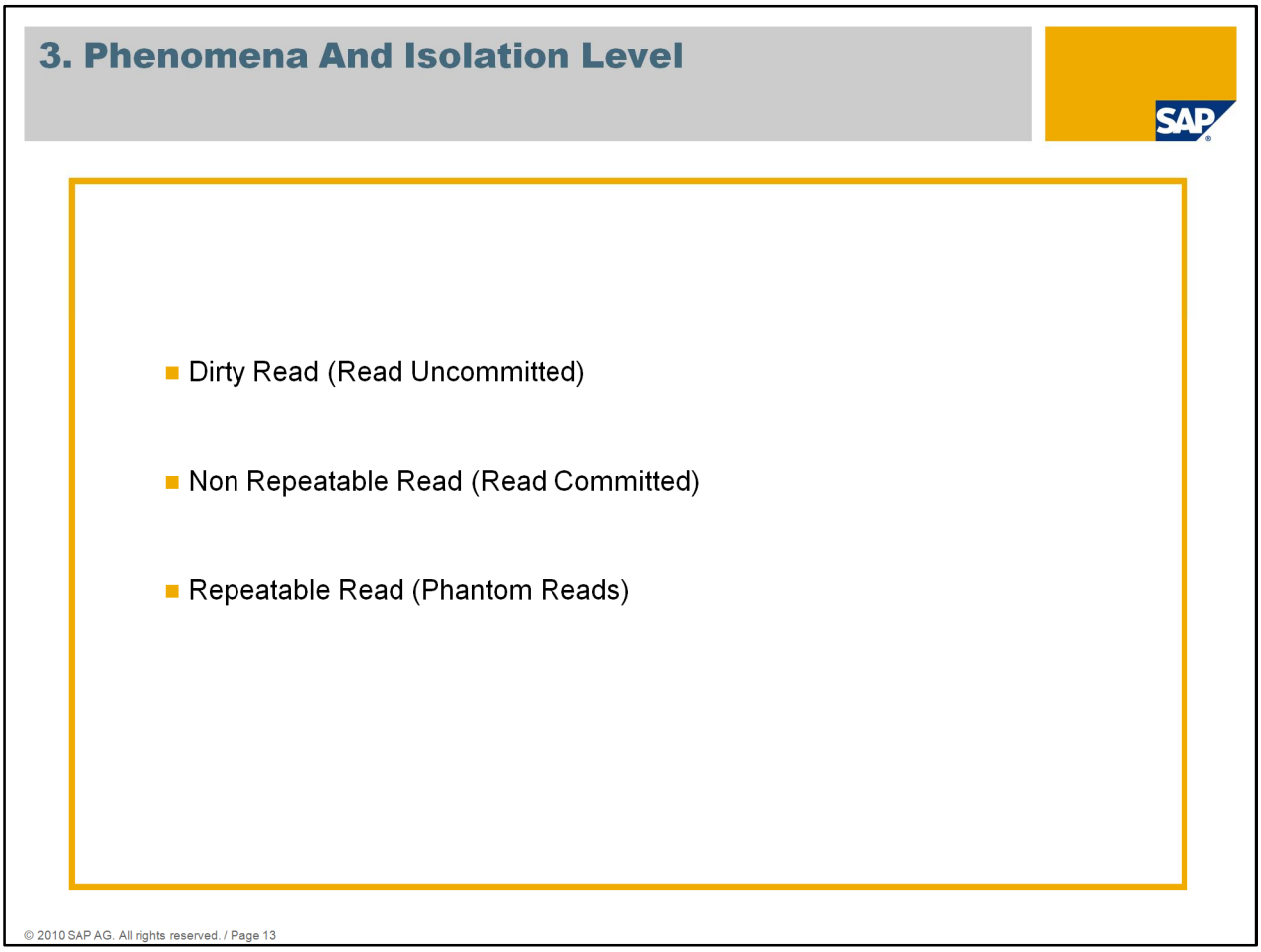
Additionally, the following applies:

- 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.

Agenda

1. Introduction
2. Lock Objects And Locking Types
3. Phenomena And Isolation Level
4. System Monitoring
5. Lock Escalation
6. Deadlock
7. Which Application Blocks The System?
8. Summary

All rights reserved.



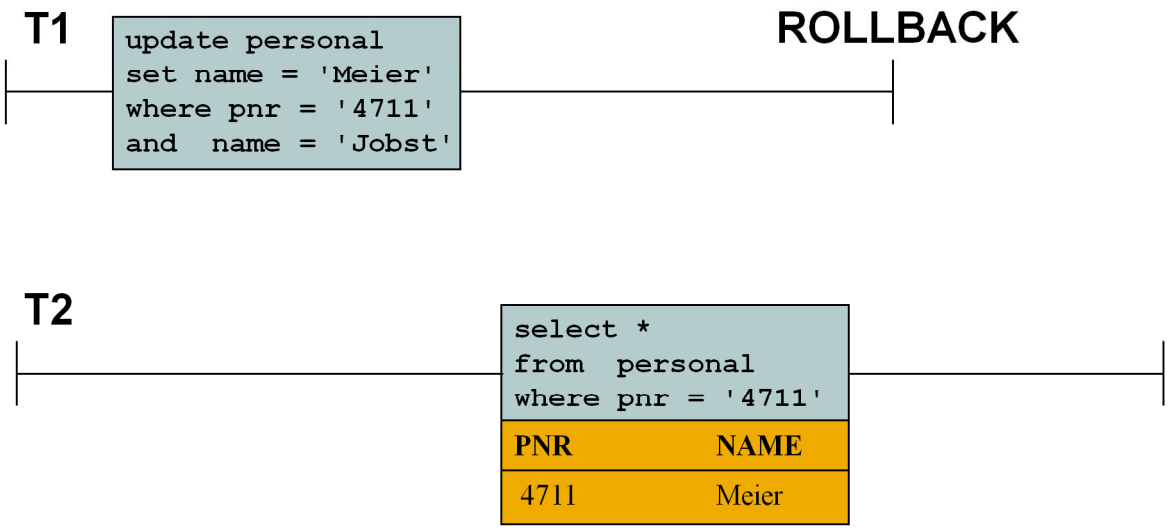
The **isolation level** plays an important role in the lock activities of the database system. You use the isolation level to specify whether locks are requested or released implicitly, and how.

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.

3. Phenomena And Isolation Level

3.1. Dirty read

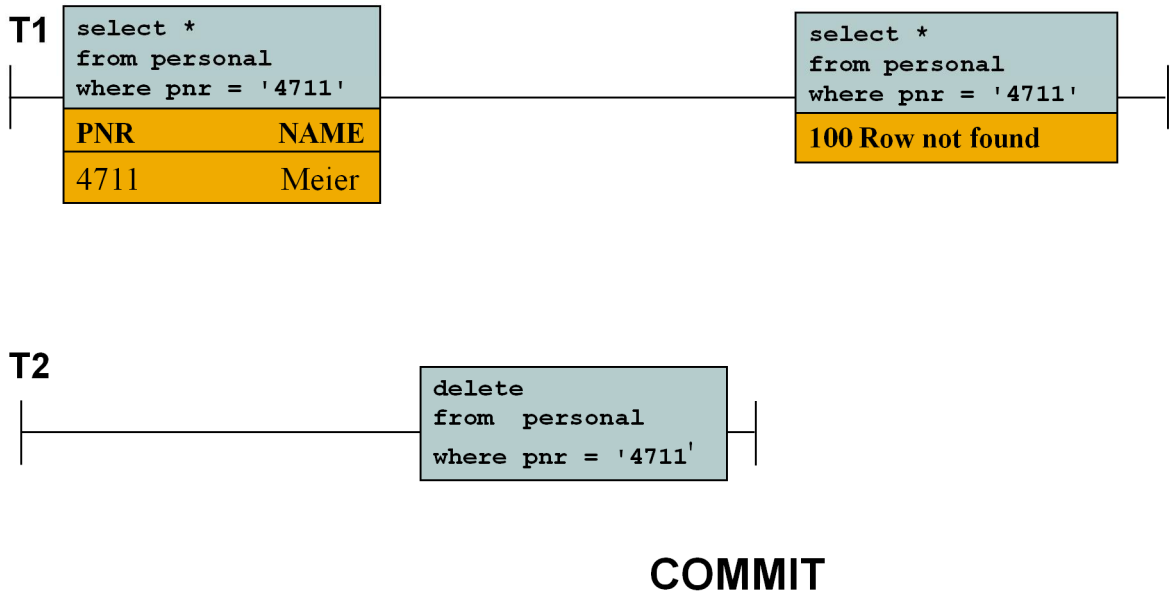


© 2010 SAP AG. All rights reserved. / Page 14

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.

3. Phenomena And Isolation Level

3.2. Non repeatable read

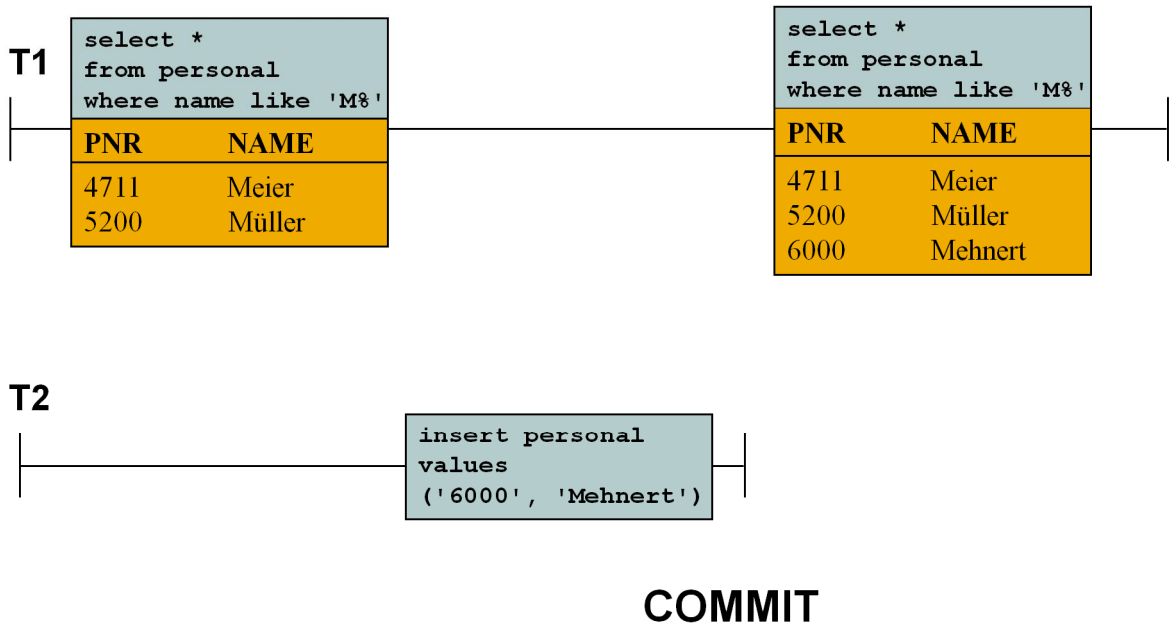


© 2010 SAP AG. All rights reserved. / Page 15

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.

3. Phenomena And Isolation Level

3.3. Phantom

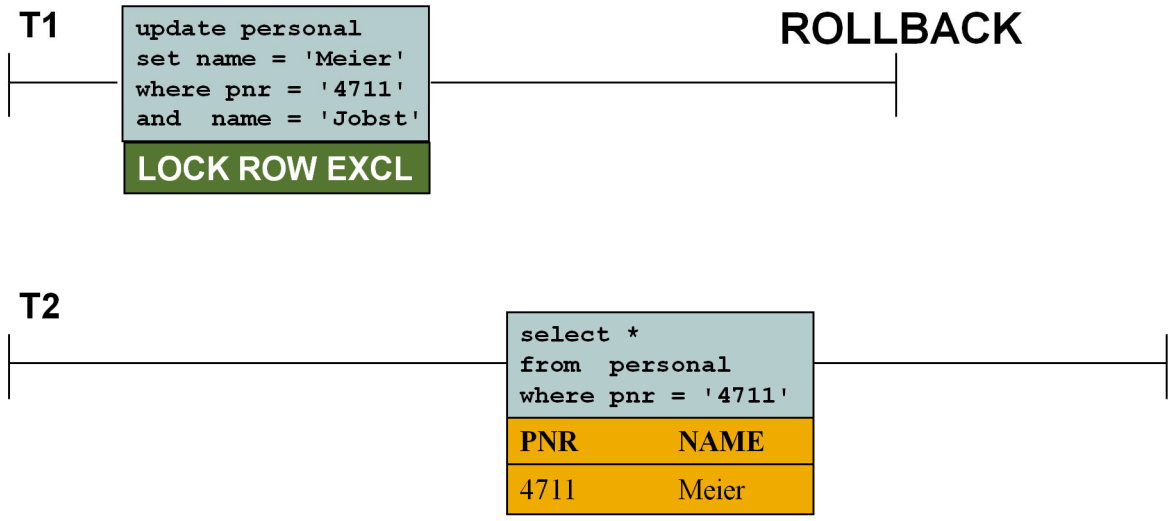


© 2010 SAP AG. All rights reserved. / Page 16

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.

3. Phenomena And Isolation Level

3.4. Isolation level 0 (read access)



© 2010 SAP AG. All rights reserved. / Page 17

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

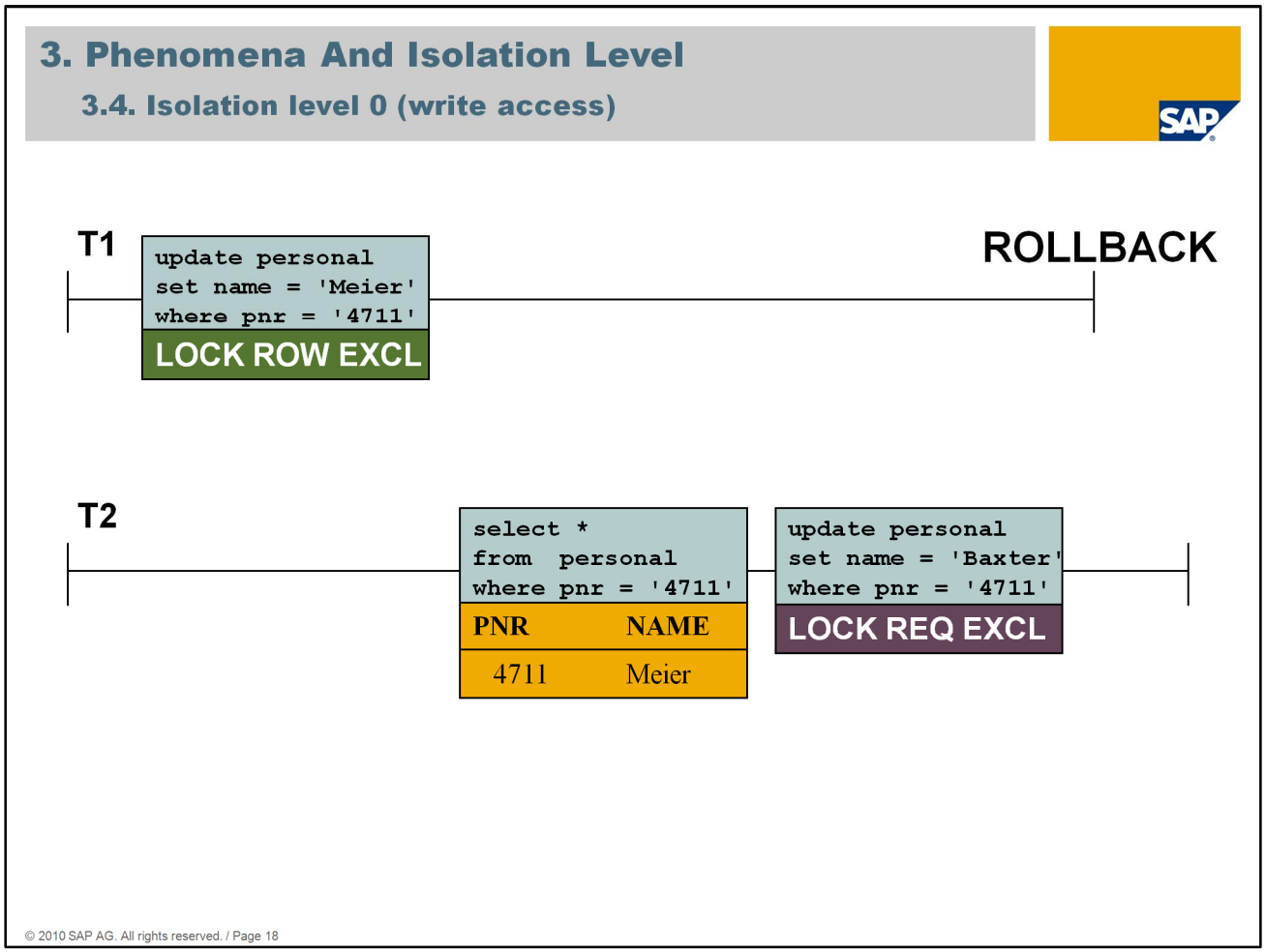
When rows are inserted, updated or deleted, **implicit exclusive locks** are assigned to the transaction for the rows affected (T1).
These cannot be released until the end of the transaction.

Read access:
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.

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)



When rows are inserted, updated or deleted, **implicit exclusive locks** are assigned to the transaction for the rows affected.
These cannot be released until the end of the transaction.

Write access:

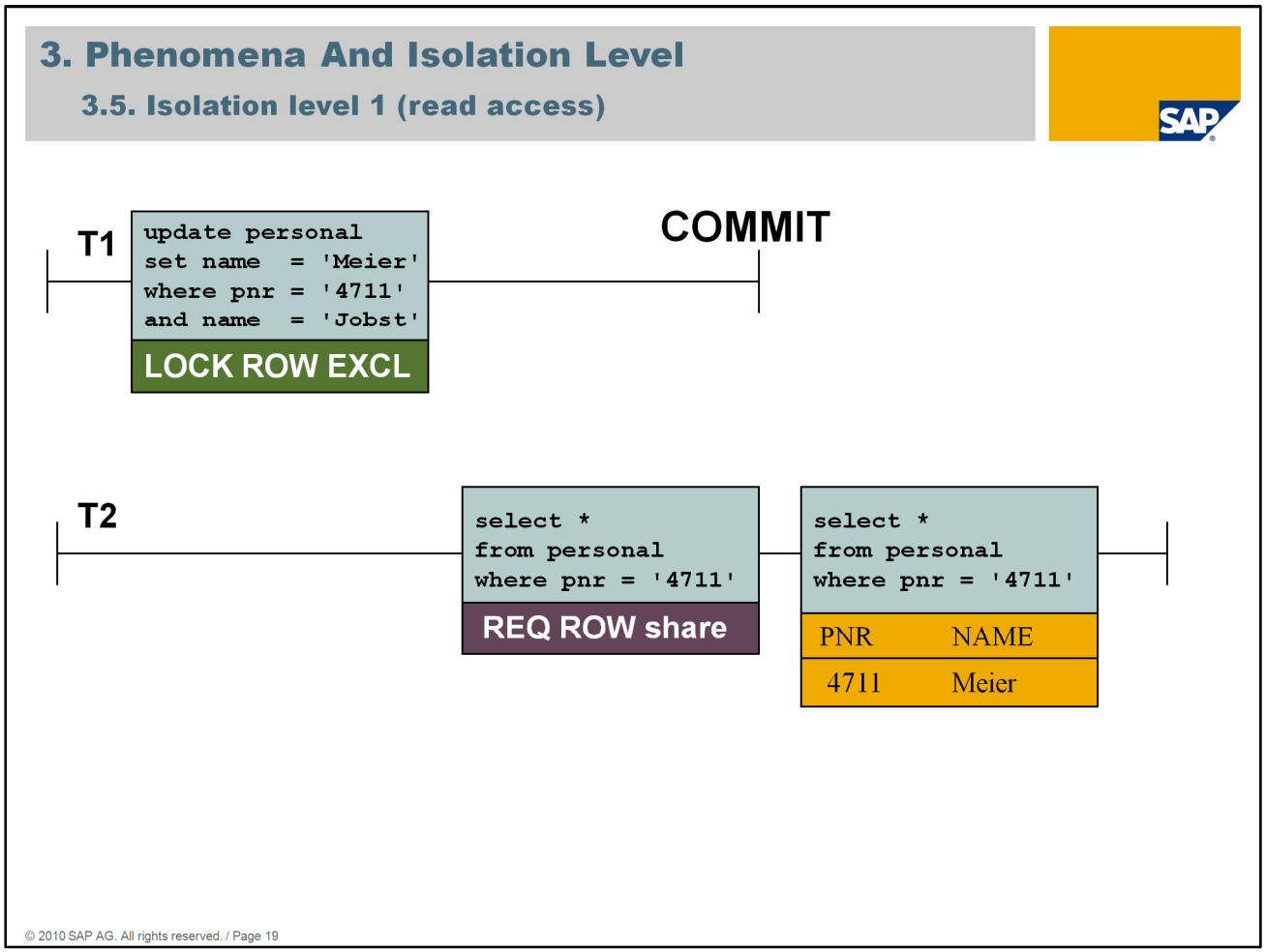
Rows have to be exclusively locked before writing (Insert, Update, Delete)

LOCK ROW EXCL -> transaction got the exclusive lock on the row (T1)
LOCK REQ EXCL -> transaction is waiting for an exclusive lock (T2)

Locks are kept until the end of transaction, so that

- no concurrent modifications can take place
- this status can be made persistent (committed)

For SAP applications (ABAP stack), the isolation level is defined in the XUSER file. The ABAP applications run at isolation level 0. To avoid the phenomena caused by isolation level 0 ABAP programming (Open SQL) offers an own locking mechanism so called enqueue lock objects.



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. As of SAP MaxDB version 7.4 the share lock is removed after the record has been read.

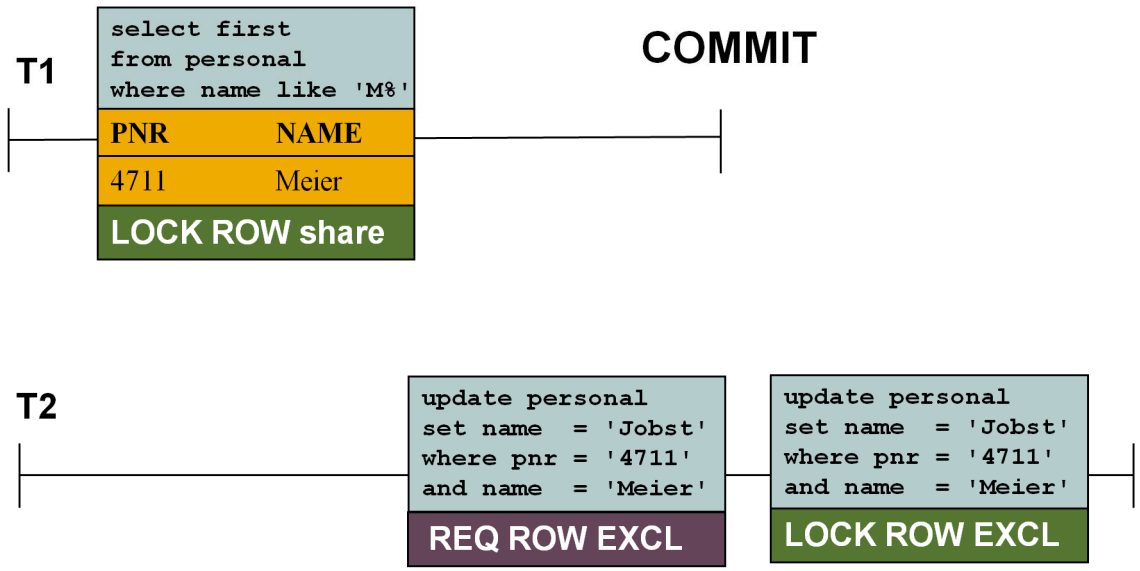
Locking of data entities and optimal multi-user operation are in direct conflict with one another. It is not recognizable whether the waiting user is waiting for a lock or whether the system is running poorly.

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.



© 2010 SAP AG. All rights reserved. / Page 20

If you specify the isolation level 1, then **a shared lock is assigned to the transaction for a read row** R1 of a table.

Transaction T2 cannot get the exclusive lock because the Share lock on row R1 (4711/Meier) has not been released.

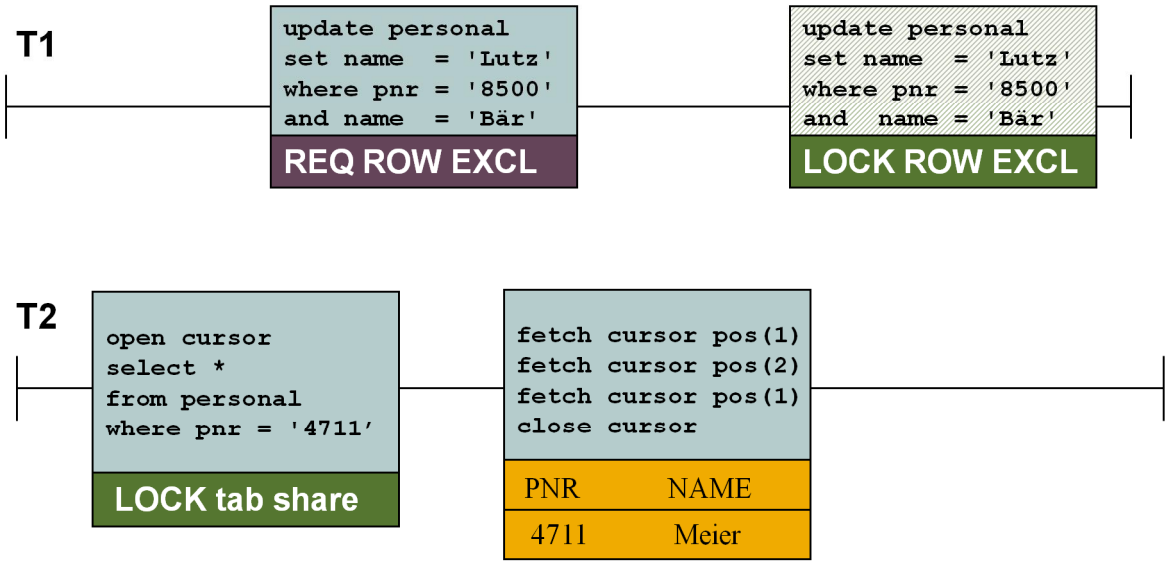
Transaction T2 requests a LOCK ROW EXCL

In NetWeaver (Java stack), data sources are used for the application logon. These data sources can be configured. You can freely choose the isolation level. If the data source does not specify an explicit isolation level, default isolation level 1 (read committed) is used.

There exists more isolation level which will be explained in the slides. In this session we won't explain those isolation level because they are not used in SAP Application environments by default.

3. Phenomena And Isolation Level

3.6. Isolation level 15 (1)



© 2010 SAP AG. All rights reserved. / Page 21

Isolation level 15 ensures that the result set does not change as long as it is being processed.

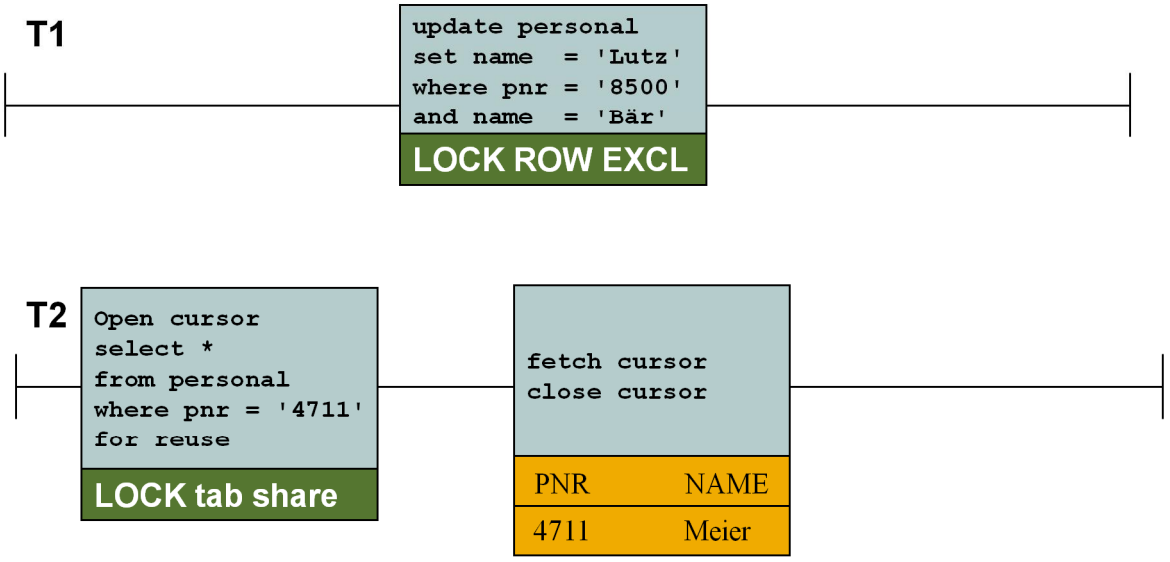
Reading backwards and positioning the pointer in the result set generates unique results.

For all SQL statements, the behavior described for isolation level 1 or 10 also applies for isolation level 15: The only difference is that with isolation level 15, shared locks are requested for all the tables addressed by the SQL statement before processing starts. If the SQL statement generates a result table, which is not physically stored, 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.

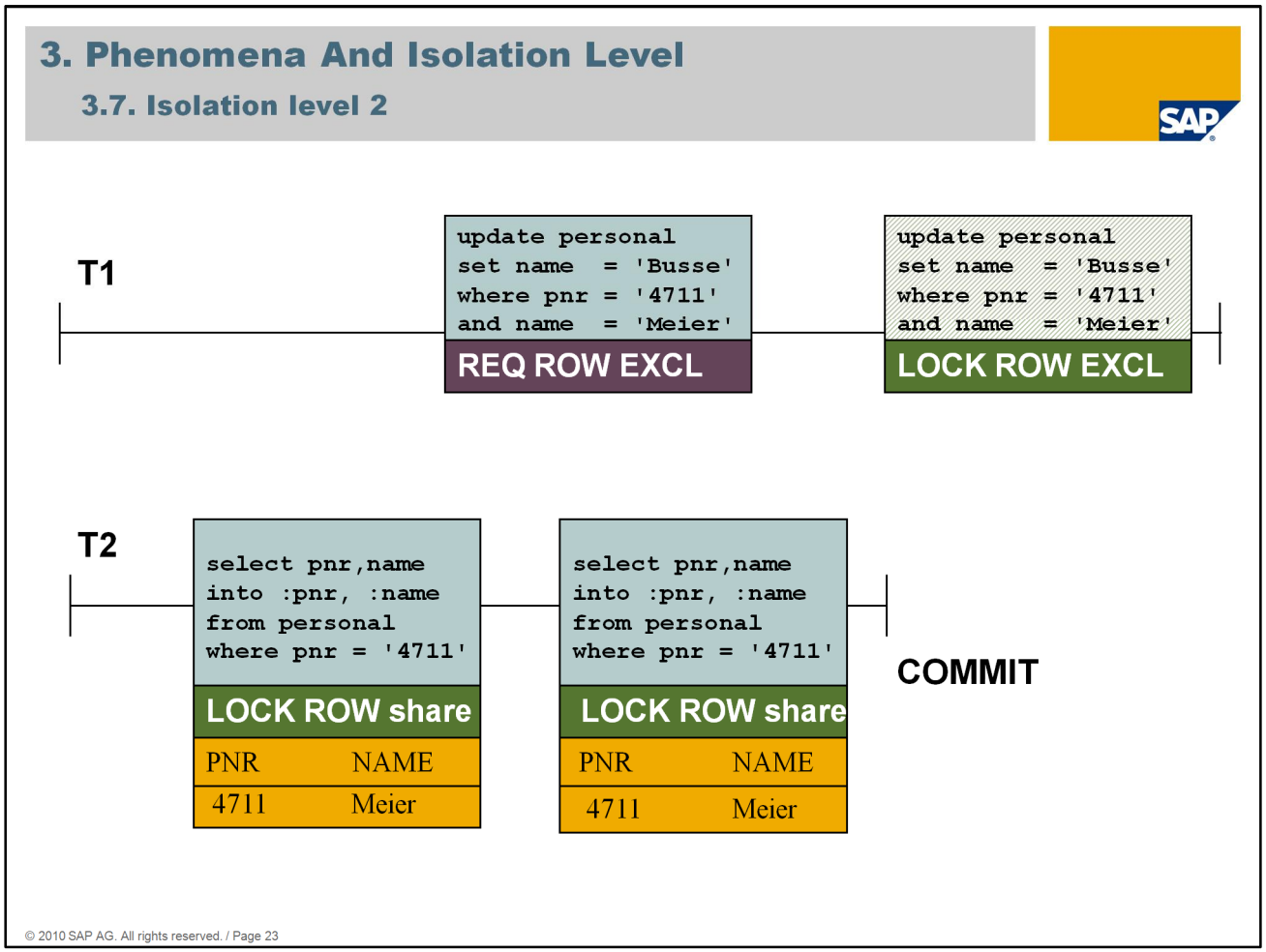
When inserting, changing or deleting rows, the exclusive locks are assigned implicitly to the transaction for the relevant rows that are not released until the end of the transaction.

3. Phenomena And Isolation level

3.6. Isolation level 15 (2)



© 2010 SAP AG. All rights reserved. / Page 22



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.

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.

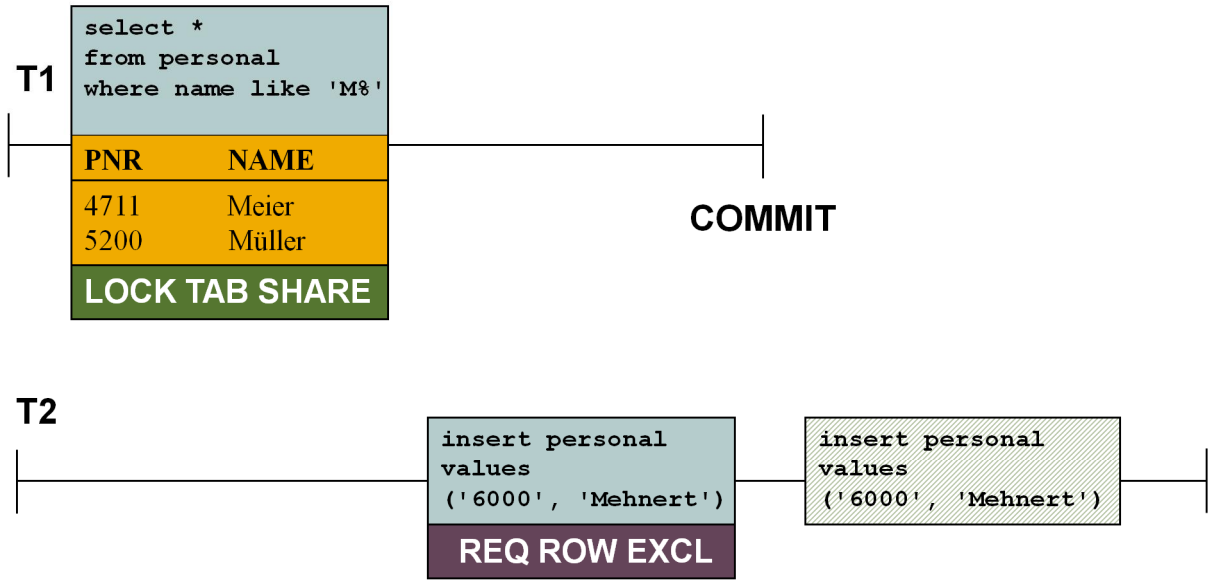
The table shared lock is not assigned to the transaction with SQL statements, where exactly one row in a table is processed that is determined by key specifications or using `CURRENT OF <result_table_name>`.

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.

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.

3. Phenomena And Isolation level

3.8. Isolation level 3 (1)



© 2010 SAP AG. All rights reserved. / Page 24

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>.`

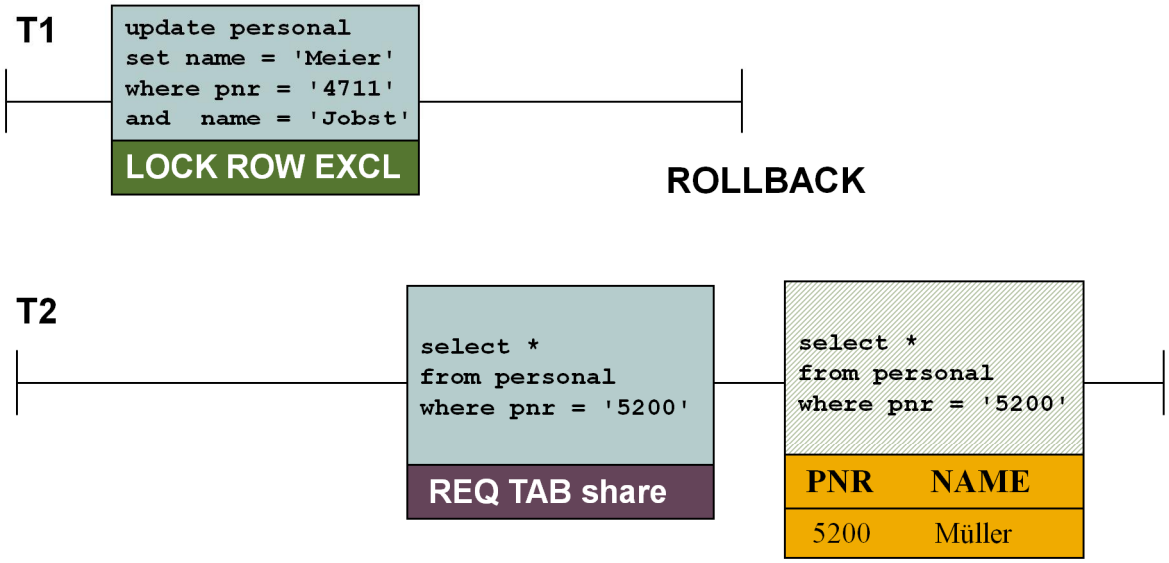
Isolation level 3 safeguards against three types of access anomalies:

- Dirty Read
- Non Repeatable Read
- Phantom

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.

3. Phenomena And Isolation level

3.8. Isolation level 3 (2)



© 2010 SAP AG. All rights reserved. / Page 25

Agenda

1. Introduction
2. Lock Objects And Locking Types
3. Phenomena And Isolation Level
4. System Monitoring
5. Lock Escalation
6. Deadlock
7. Which Application Blocks The System?
8. Summary

All rights reserved.

4. System Monitoring

DBACOCKPIT (DB50)

- SQL lock overview and waiting status

System tables and views

- sysdba.lockstatistics
- sysdba.locliststatistics
- sysdba.transactions
- domain.locks
- domain.lock_holder and domain.lock_requestor
- domain.lock_waits

Database console

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

© 2010 SAP AG. All rights reserved. / Page 27

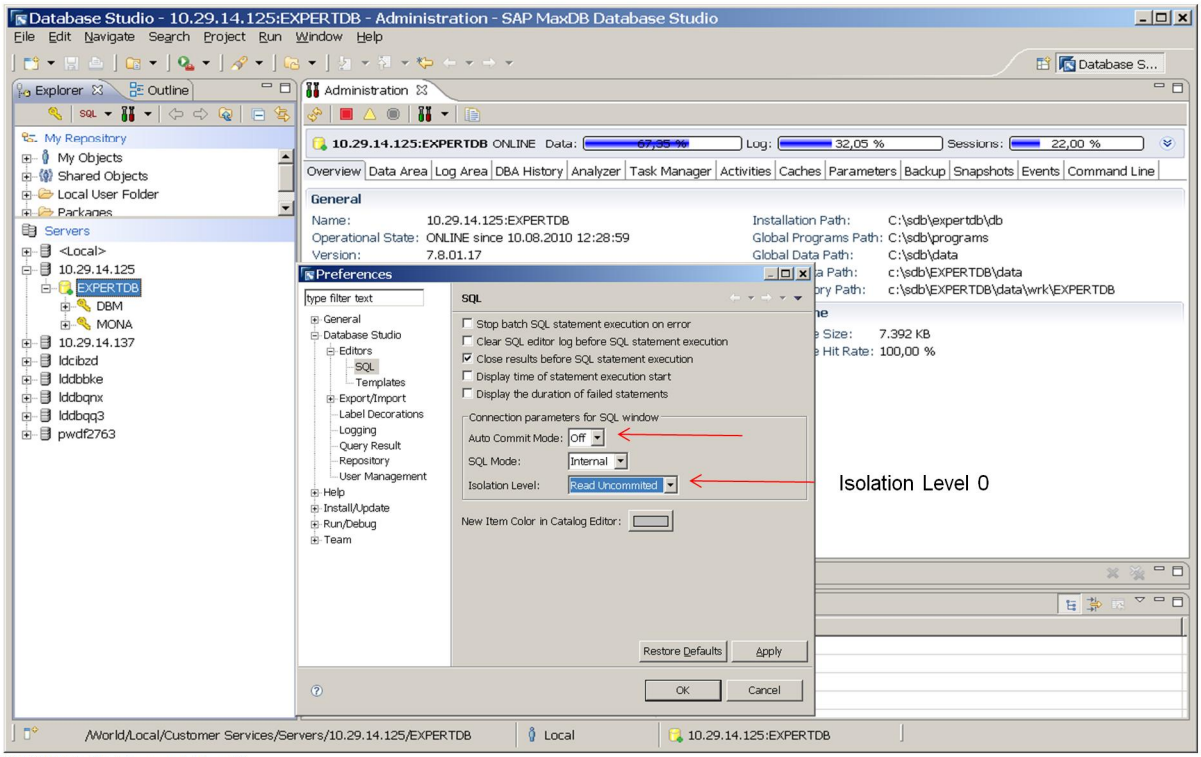
For system monitoring transaction *DBACockpit* (or *DB50* in older SAP releases) can be used if an ABAP Stack or a Solution Manager is available.

MaxDB system tables can also be used for analysing lock situations but is less comfortable than using the transaction described above.

The database console program x_cons can be directly used on OS level or in transaction *DBACockpit*. This tool is used for more detailed analysis directly on the database. Tasks which are holding or requesting locks are shown in status *Vwait*.

4. System Monitoring

4.1. Monitoring lock situations: Demo environment



© 2010 SAP AG. All rights reserved. / Page 28

Two of three examples in this demo session can be reproduced using the SAP MaxDB tutorial, which is available with each SAP MaxDB Software version. The toutorial can be loaded with *Database Studio -> Administration Tasks –> Load Tutorial*.

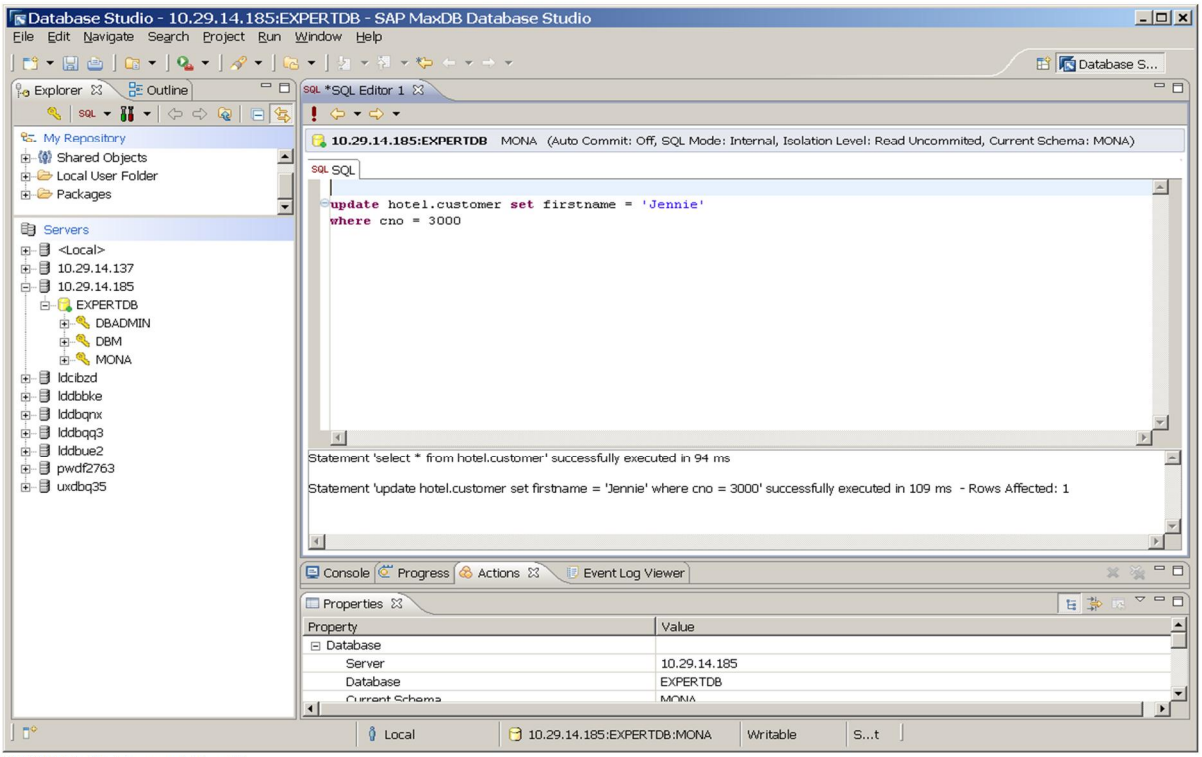
To reproduce the demo examples you have to change the Database Studio preferences as follows:
Start Database Studio and choose:
Window -> Preferences -> Database Studio -> Editors -> SQL
Automcommit has to be switched *off.*

Isolation level must be set to *Read Uncommitted* (Dirty Read).

After these changes were applied each new SQL Editor inherits this environment.

4. System Monitoring

4.1. Monitoring lock situations: Transaction 1: Update ...



© 2010 SAP AG. All rights reserved. / Page 29

To create a lock situation a minimum of 2 transactions are necessary.
For the first example in this expert session we are using table *hotel.customer:*

select * from hotel.customer

With the following update command the first name of customer with cno= 3000 will be changed from Jenny to Jennie.

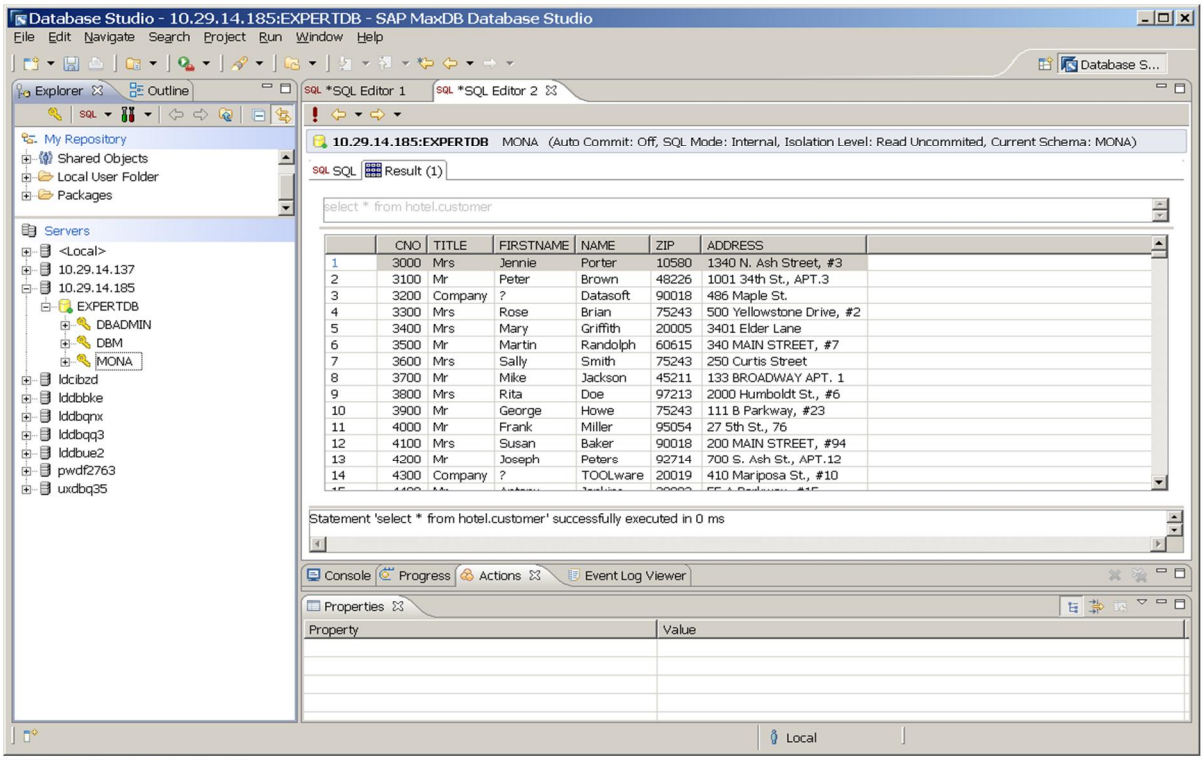
update hotel.customer set firstname = 'Jennie' where cno = 3000

This SQL statement is not commited. The system is running in isolation level 0.

In the next slide we check which value for firstname is displayed by the second transaction.

4. System Monitoring

4.1. Monitoring lock situations: Transaction 2: Select ...



© 2010 SAP AG. All rights reserved. / Page 30

We open a second transaction (Editor 2) and check the content of table *hotel.customer*.
Remember that Transaction 1 is still not committed or rolled back.

select * from hotel.customer

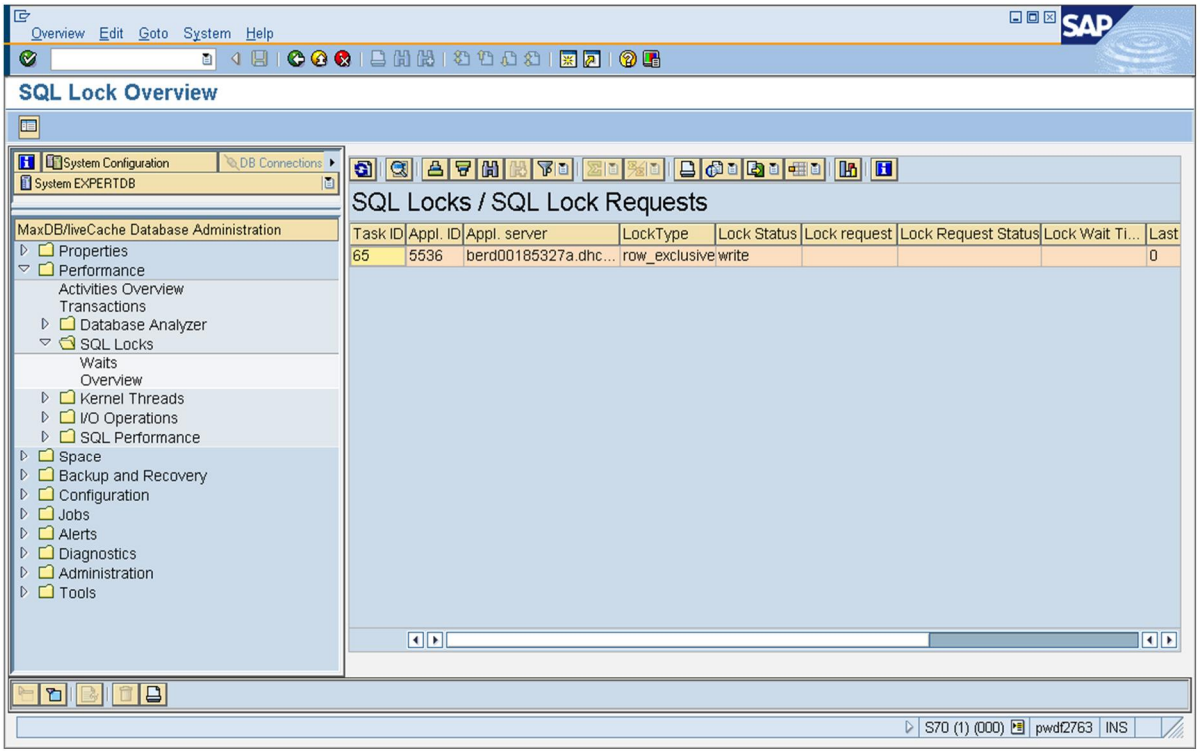
The select is working no share lock has to be set in isolation level 0.

The system is reading uncommitted therefore we already see the changed value of firstname = ‚Jennie‘

Next step is to check the lock overview in transaction *DBACOCKPIT* (see next slide)

4. System Monitoring

4.1. Monitoring lock situations: DBACOCKPIT Lock Overview



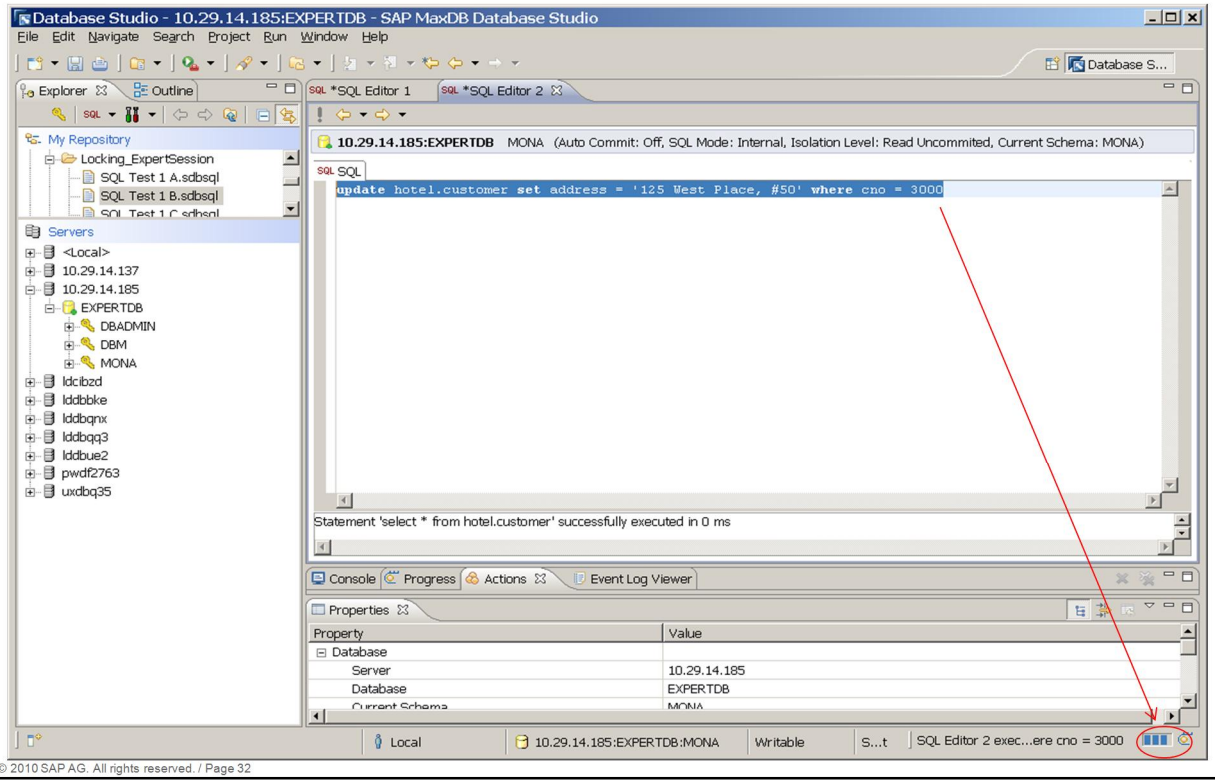
© 2010 SAP AG. All rights reserved. / Page 31

In transaction *DBACOCKPIT* we can use *Performance -> SQL Locks -> Overview* to check which and how many SQL locks are currently existing.

Task ID 65 holds an exclusive lock on one row of the table *customer*. There is no other transaction which requests a lock on the same row (column *Lock Request* is empty).

4. System Monitoring

4.1. Monitoring lock situations: Transaction 2: Update ...



© 2010 SAP AG. All rights reserved. / Page 32

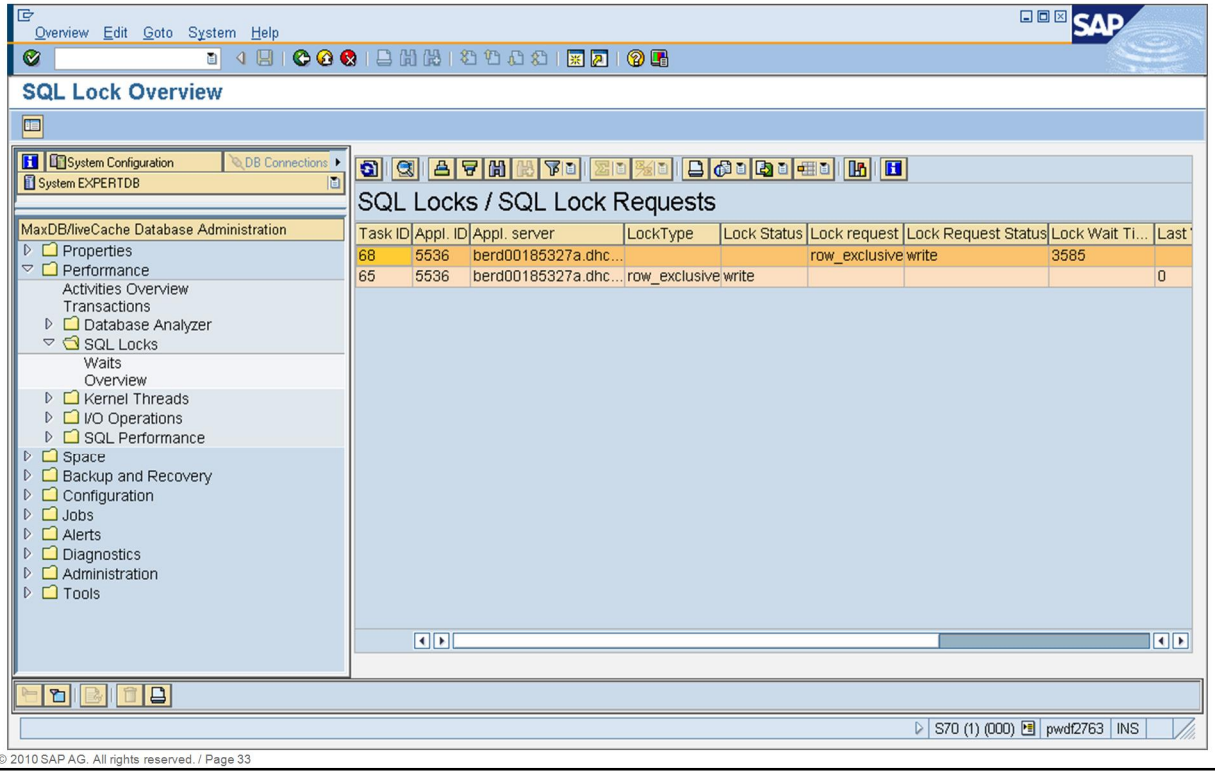
To create a lock wait situation in isolation level 0 a minimum of 2 transactions, which want to change the same record are necessary.
The second transaction wants to update the same customer cno= 3000 with a new address.

update hotel.customer set address = '125 West Place, #50' where cno = 3000

The update statement is hanging in Database Studio …

4. System Monitoring

4.1. Monitoring lock situations: DBACOCKPIT Lock Overview



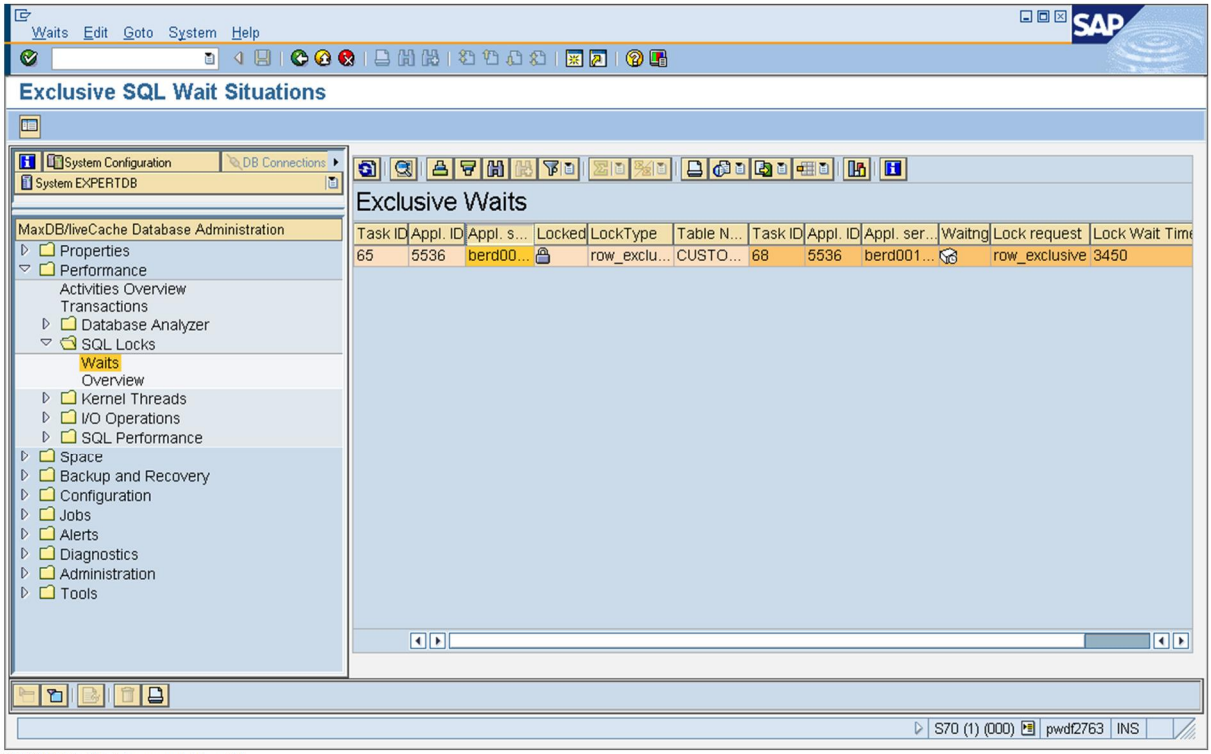
Each transaction which wants to change records has to set locks on the records. In this example we have 2 transactions which want to change records. We have 2 entries in the lock overview.

Task ID 68 (which is the second SQL session which wants to change the address) wants to have an exclusive lock (*Lock request*).
Task ID 65 (which is the first transaction which changes the first name) has the exclusive lock on the record with cno=3000.

To find out if those transactions are both trying to change the same records the *DBACOCKPIT -> Performance -> SQL Locks -> Waits* has to be used.

4. System Monitoring

4.1. Monitoring lock situations: DBACOCKPIT Lock Waits



© 2010 SAP AG. All rights reserved. / Page 34

In transaction *DBACOCKPIT -> Performance -> SQL Locks -> Waits* the system shows which transactions are holding locks and which transactions are requesting those locks.

In the example of this session:

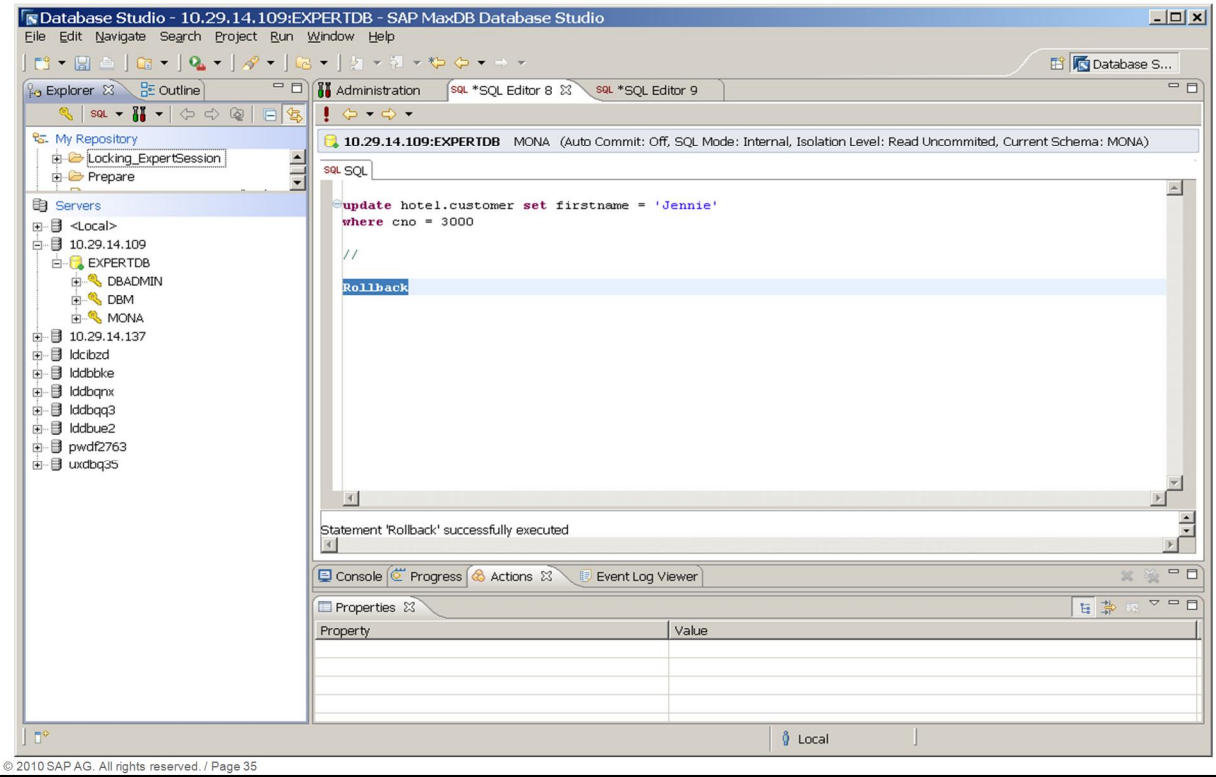
- Task ID 65 has a row_exclusive lock (*Lock Type*) on table *customer*
- Task ID 68 is requesting a row_exclusive lock (*Lock request*) on the same record

Task ID 68 is in status waiting (*Waiting*). The *Lock Wait Time* displays the remaining wait time for this lock (in seconds). Initial lock wait time is the value of parameter **RequestTimeout**. The database parameter RequestTimeout defines how long a transaction is waiting at a max for the assignment of a lock. If this wait time has expired before the lock is assigned, the transaction terminates with an error *‚-51 Lock request timeout*'.

TaskID 68 will get the lock as soon as Task ID 65 gave the lock free again (after COMMIT or ROLLBACK). If Task ID 65 does not give the lock free again in the remaining lock request wait time, Task ID 68 will get an error message *‚-51 Lock request timeout'.*

4. System Monitoring

4.1. Monitoring lock situations: Transaction 1: Commit/Rollback

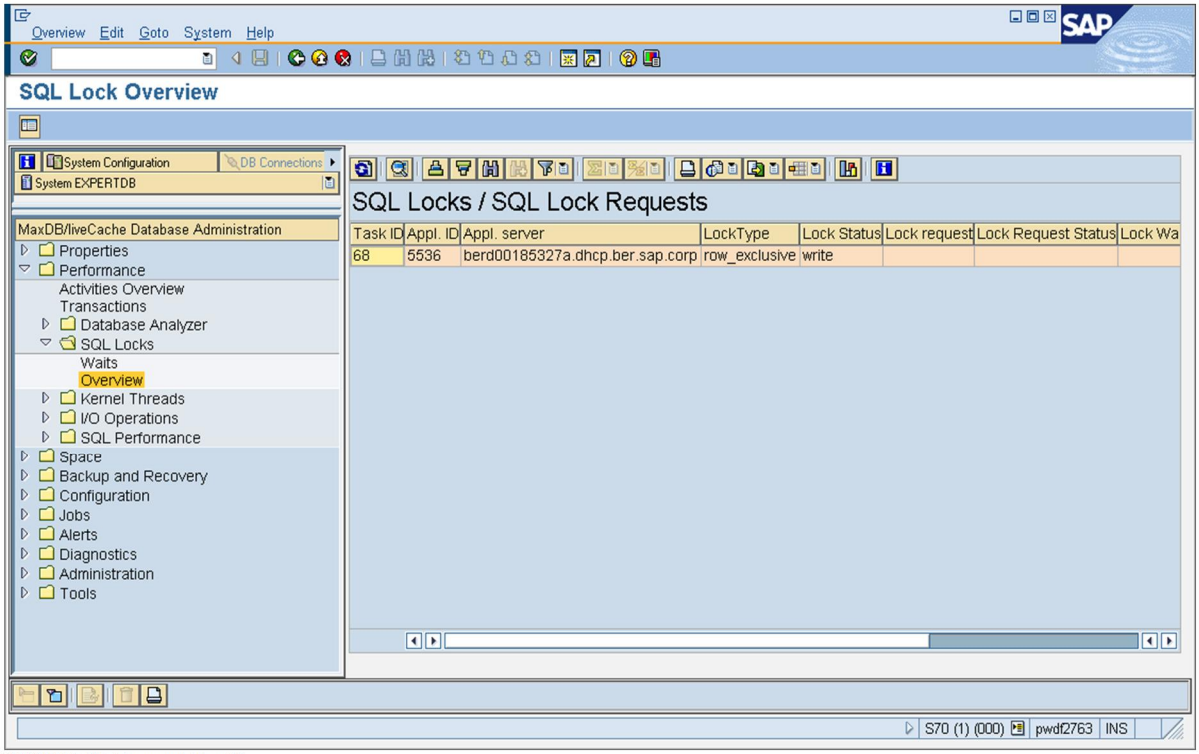


© 2010 SAP AG. All rights reserved. / Page 35

When transaction 1 will be finished (COMMIT or ROLLBACK) the exclusive lock is released.

4. System Monitoring

4.1. Monitoring lock situations: DBACOCKPIT Lock Overview

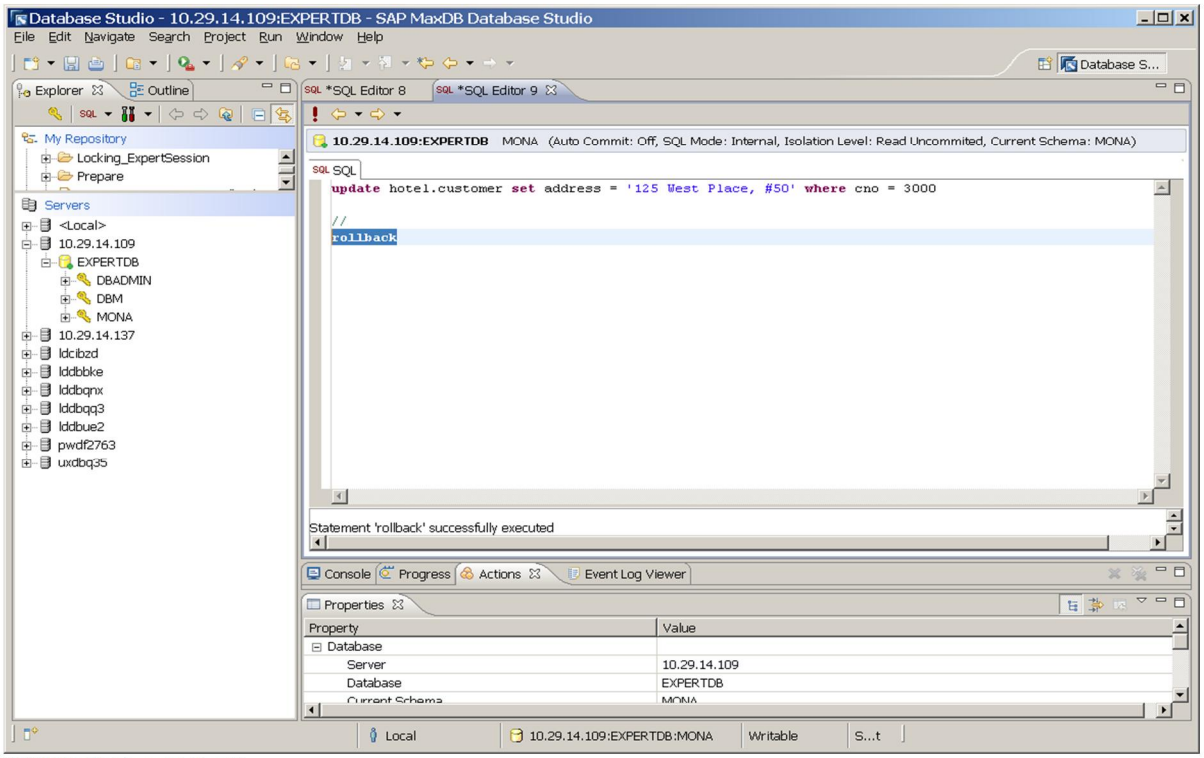


© 2010 SAP AG. All rights reserved. / Page 36

After the exclusive lock is released transaction 2 (Task ID 68) got the exclusive lock. This lock persists until transaction 2 finishs with ROLLBACK or COMMIT.

4. System Monitoring

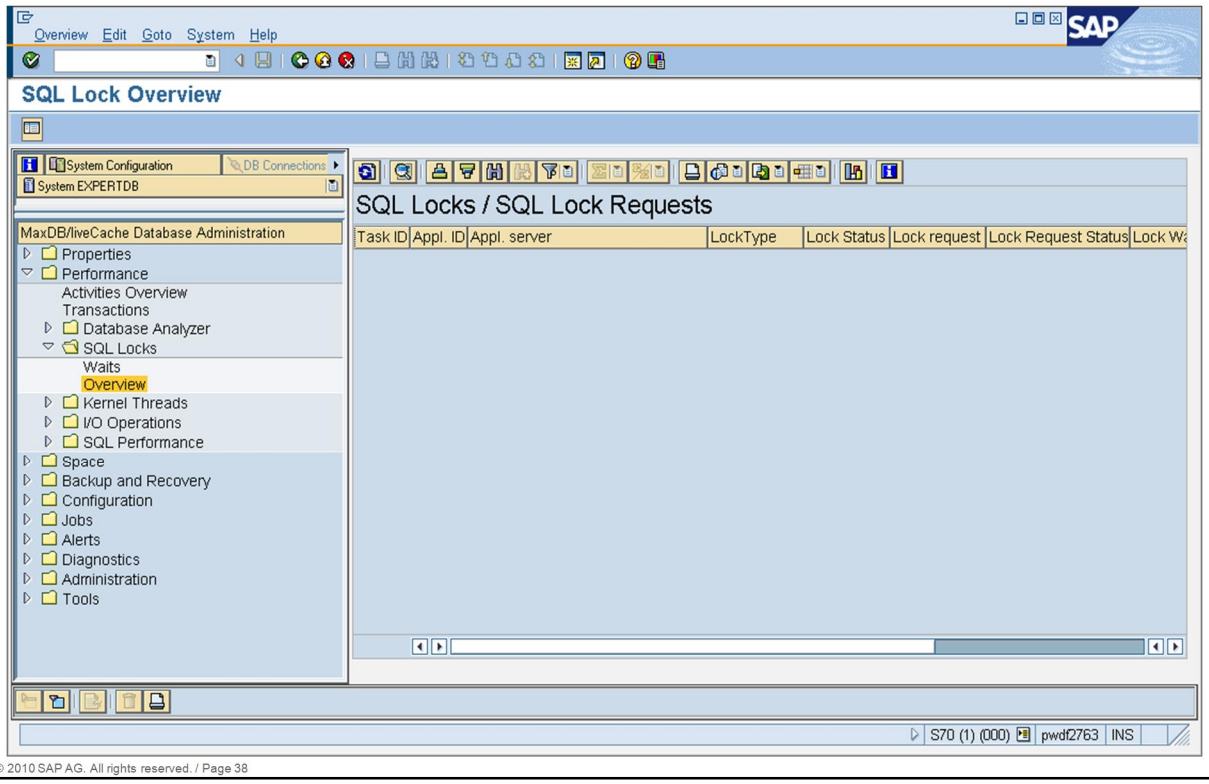
4.1. Monitoring lock situations: Transaction 2: Commit/Rollback



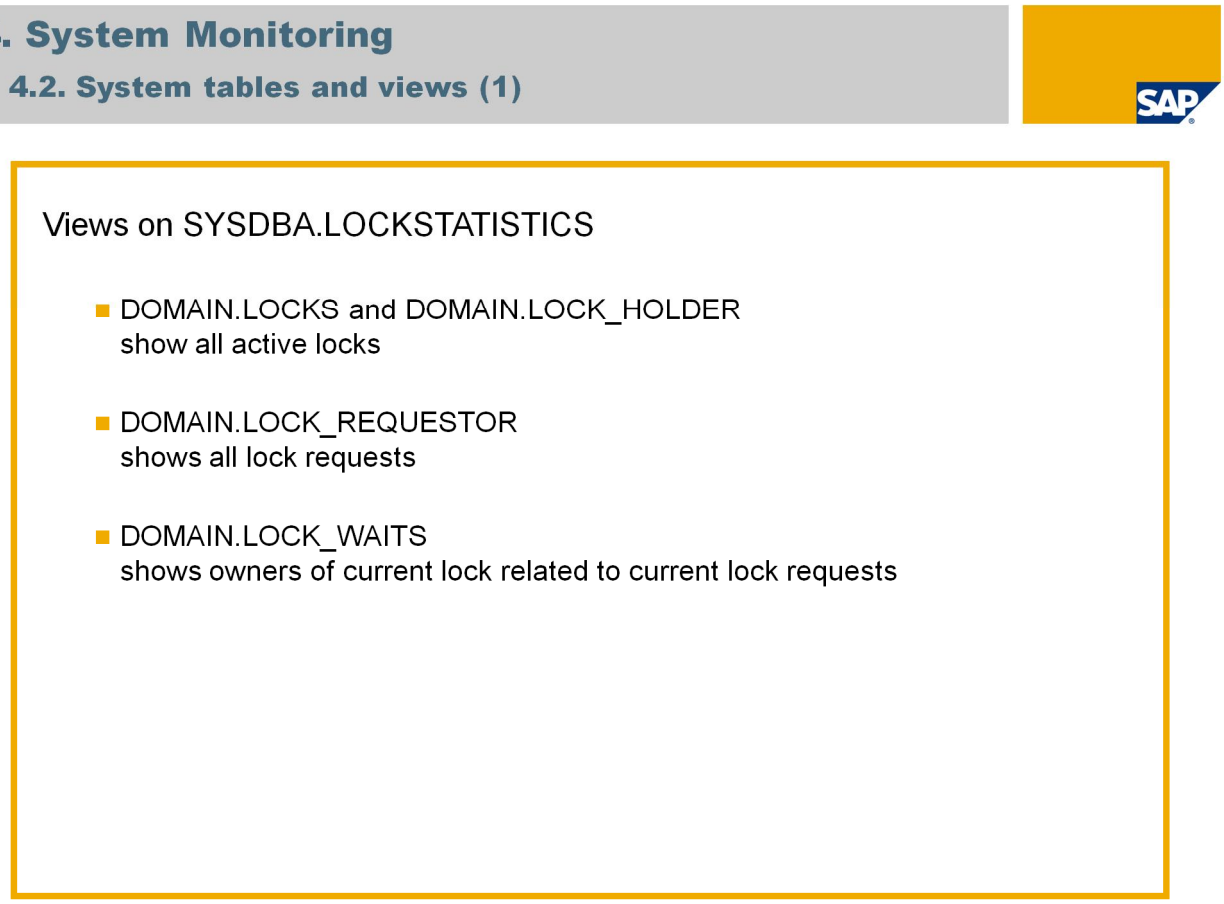
© 2010 SAP AG. All rights reserved.

4. System Monitoring

4.1. Monitoring lock situations: DBACOCKPIT Lock Overview



No locks exists anymore when both transactions in the database EXPERTDB have been closed with COMMIT or ROLLBACK.



© 2010 SAP AG. All rights reserved. / Page 39

To monitor lock situations you can use transaction *DB50* or *DBACOCKPIT*.

When you are working in an NON-SAP-Environment you can use the SAP MaxDB system tables to monitor lock situations.

DOMAIN.LOCKS or DOMAIN.LOCK_HOLDER are showing all locks which currently exist in the system.

The output of these system tables is shown in the transaction *DBACOCKPIT –> Performance -> SQL Locks –> Overview*.

System table DOMAIN.LOCK_REQUESTOR lists all locks which are currently requested.

System table DOMAIN.LOCK_WAITS lists all owners of current locks related to current lock requests.
The output of this system table is shown in the transaction *DBACOCKPIT –> Performance -> SQL Locks –> Overview*.

SYSDBA.LOCKSTATISTICS

SESSION	internal session id
TRANSACTION	internal transaction id
PROCESS	task id of bound kernel task
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
OWNER	owner of table
TABLENAME	name of table
ROWIDLENGTH	length of locked key
ROWID	locked key
ROWIDHEX	hexadecimal representation of locked key
…	

© 2010 SAP AG. All rights reserved. / Page 40

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.

SYSDBA.LOCKLISTSTATISTICS

- maximum number of lock entries as defined for lock list
- 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

© 2010 SAP AG. All rights reserved. / Page 41

You will find a table description of all columns in the system table manual.

The following only lists particular columns:

- MAXLOCKS 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)
- 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

Agenda

1. Introduction
2. Lock Objects And Locking Types
3. Phenomena And Isolation Level
4. System Monitoring
5. Lock Escalation
6. Deadlock
7. Which Application Blocks The System?
8. Summary

All rights reserved.

5. Lock Escalation

5.1. What is a 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

© 2010 SAP AG. All rights reserved. / Page 43

What is a lock escalation?
A lock escalation is the conversion of line locks into a table lock.

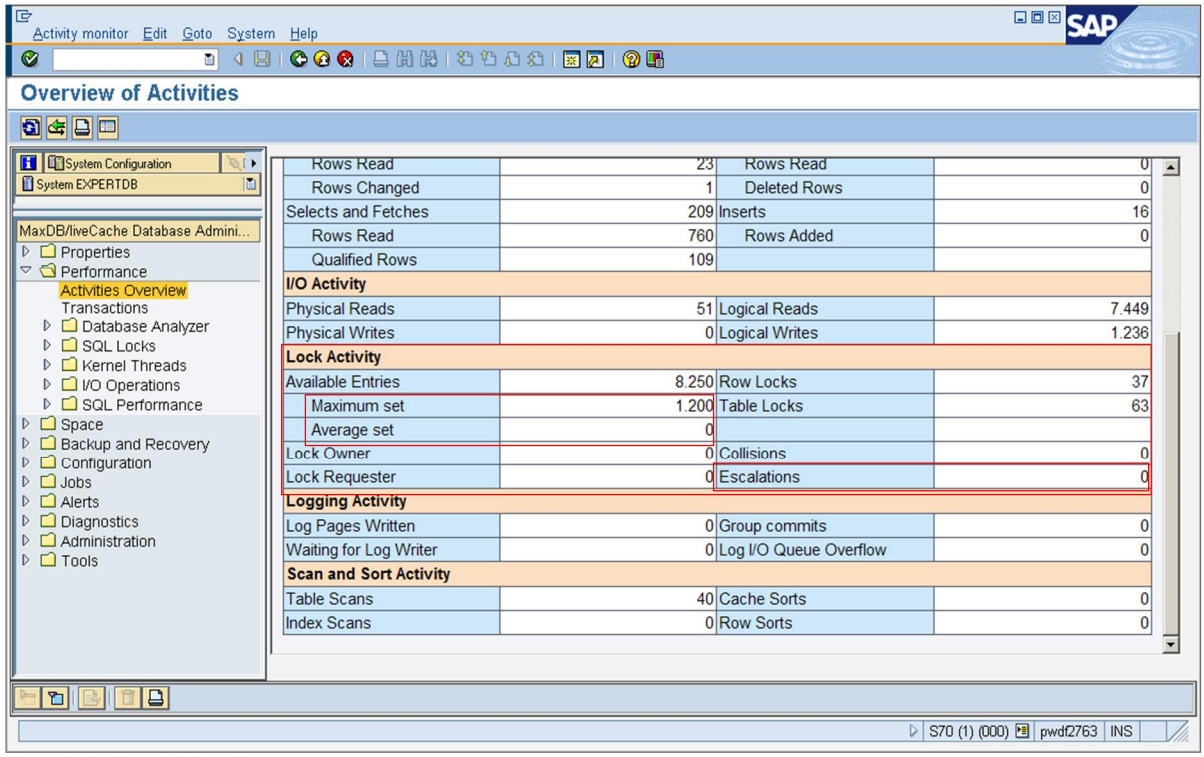
The MaxDB parameter **MaxSQLLocks** specifies the maximum size of the lock entry list. In this list, record locks and table locks of all users as well as their lock requests are managed. **If the record locks of one transaction take up more than 20% of the lock list for a table, the system carries out a lock escalation**. In this process, the record locks are converted to a table lock.

As of Version 7.6.03, an additional threshold (parameter **RowLocksPerTransaction**) was introduced for lock escalations. **If a transaction fills more than 50% of the lock list with line locks, the lock escalation is carried out for the table whose record is to be locked when this threshold value is exceeded**. The default value of parameter RowLocksPerTransaction is 50 %.

The system carries out lock escalations to prevent an overflow in the lock entry list. Table locks may affect concurrent users who have lock requests.

5. Lock Escalation

5.2. DBACOCKPIT Activities Overview



© 2010 SAP AG. All rights reserved. / Page 44

You can increase the number of possible record locks for each user and table by increasing the value of the parameter **MaxSQLLocks**.

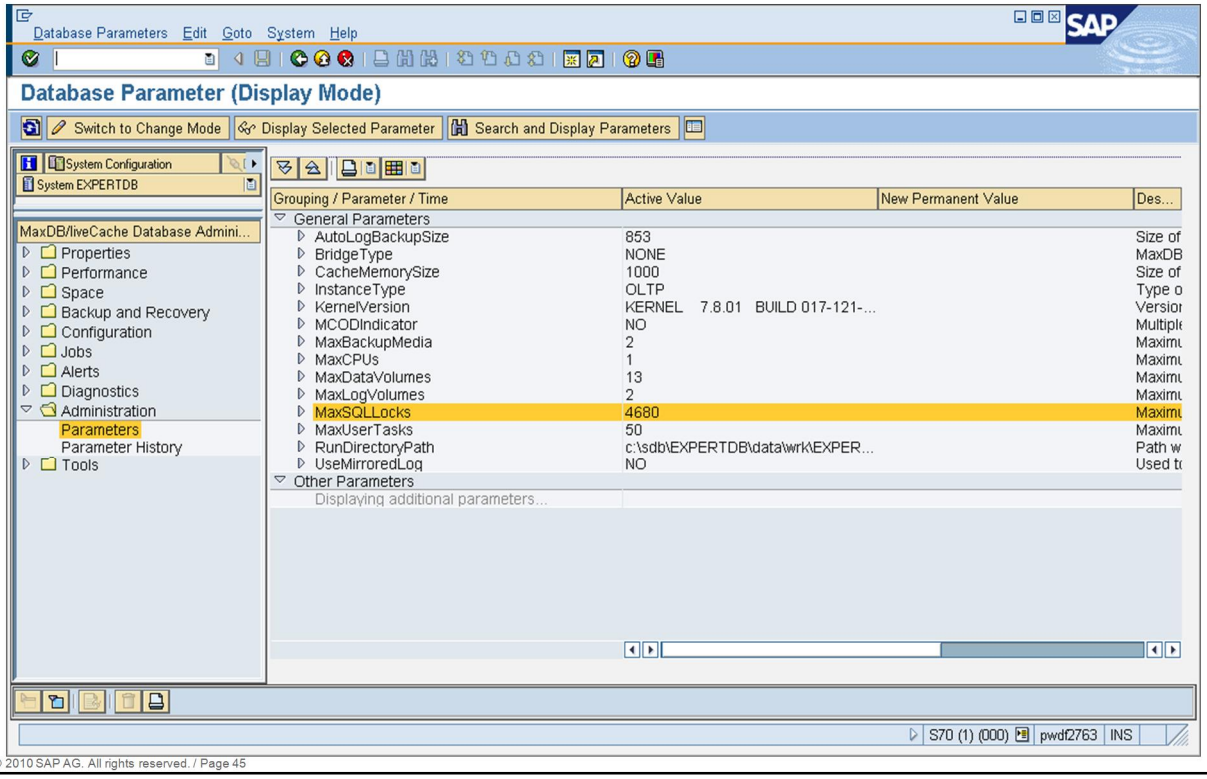
You should increase the value of MaxSQLLocks if in transaction *DBACockpit -> Performance -> Activities Overview*
Lock Activity shows:

- Available Entries -> Average set is similar to the value for MaxSQLLocks
- Available Entries -> Maximum set is identical with the value for MaxSQLLocks
- Escalations is higher than zero

Notice: The values are initialized with each shutdown of the database.

5. Lock Escalation

5.3. Parameter MaxSQLLocks



For each lock that has been defined using **MaxSQLLocks** the database takes up 200 to 300 bytes of main memory during the restart. The size of the entries depends on the database versions and the operating system. If you want to determine the exact memory consumption use the original value for MaxSQLLocks to start the database. Use the operating system to determine the memory consumption of the database process. Set MaxSQLLocks to the planned new value and set the database to offline status and then to online status. Use the operating system again to determine the memory consumption and calculate the difference between the key figures determined.

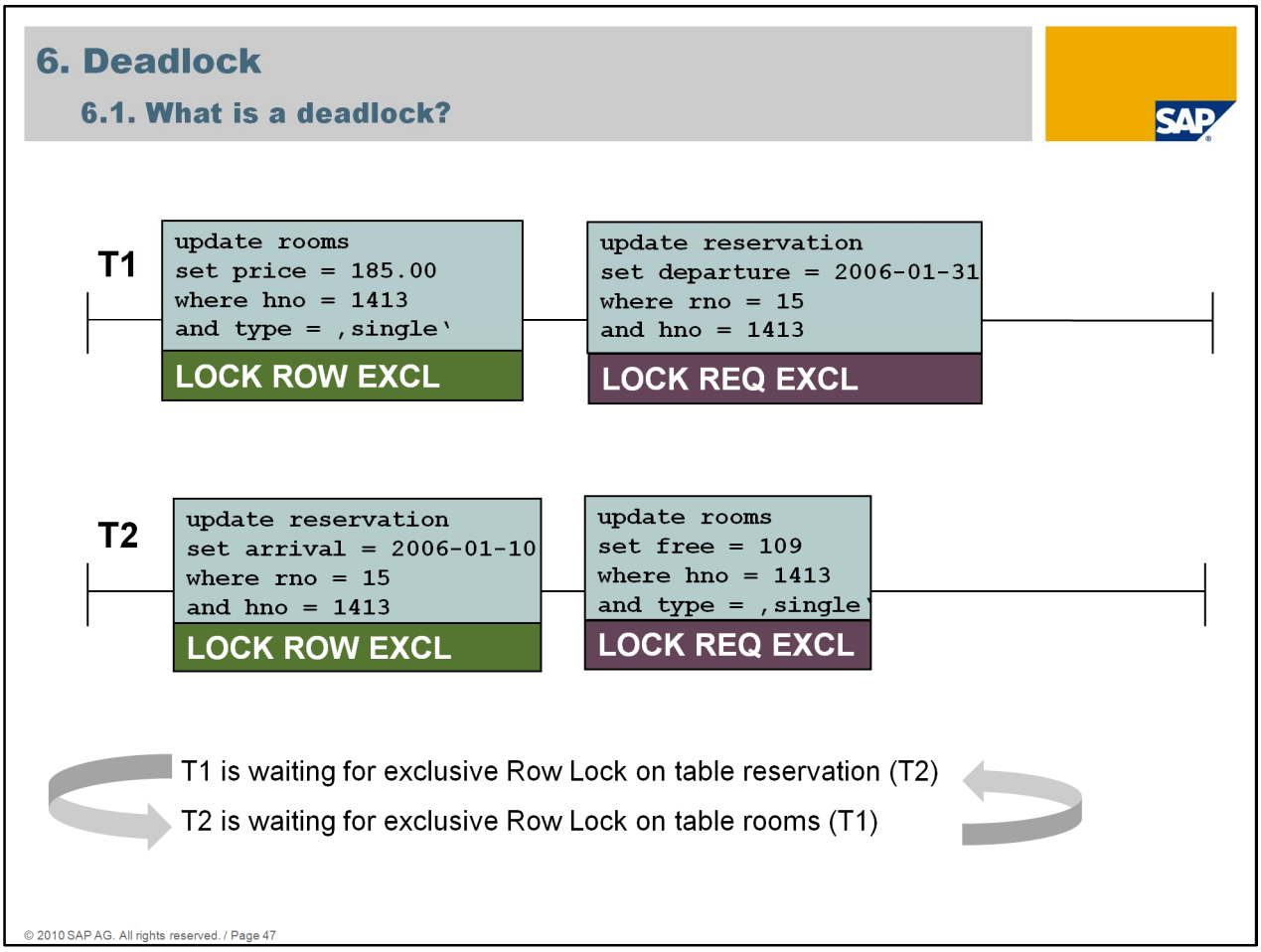
In SAP environment the values of MaxSQLLocks are between 500.000 and 1.500.000.

In Business Warehouse systems values of 2.500.000 are not seldom.

Agenda

1. Introduction
2. Lock Objects And Locking Types
3. Phenomena And Isolation Level
4. System Monitoring
5. Lock Escalation
6. Deadlock
7. Which Application Blocks The System?
8. Summary

All rights reserved.



A deadlock occurs if two or more users impede each other due to set locks and can no longer work as a result.

Two or more transactions hold locks and request further locks that are already held by the other transaction.

Deadlocks are detected by the database up to a certain depth. The system issues an error message for the user who triggers the deadlock. The deadlock is removed.

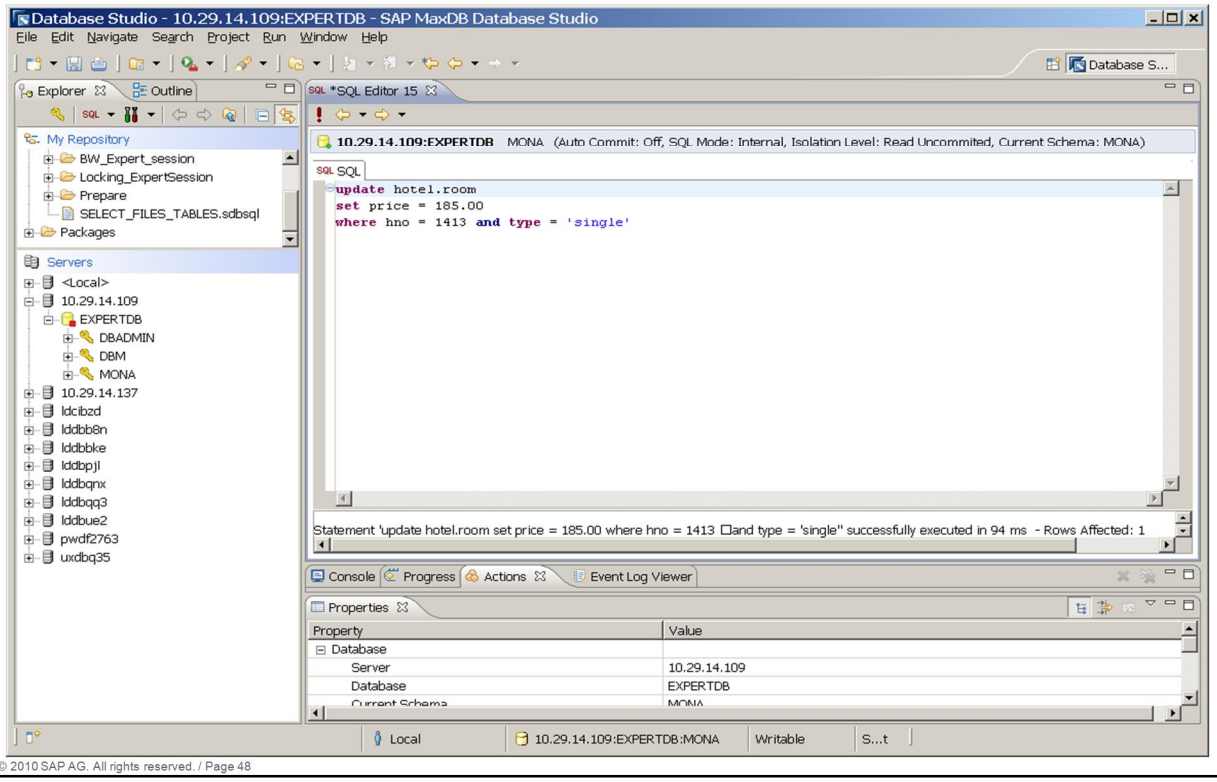
The database cannot recognize all deadlocks, for example, if shared locks are involved. A deadlock that is not detected by the system is removed by timeouts.

The database parameter **DeadlockDetectionLevel** defines the maximum depth of search for the deadlock detection with SQL locks.

If this database parameter is set to value 0, the deadlock detection is deactivated (in other words, deadlocks are removed only by **RequestTimeout**). A value that is higher than 0 defines the depth of search. Up to the specified depth of search, deadlocks are detected and immediately removed. The initial value for the deadlock detection is 4. A higher value results in considerable costs. Consider using a higher value only if an application triggers serious deadlock problems that cannot be solved in the application.

6. Deadlock

6.2. Demo with DeadlockDetectionLevel = 4 : Transaction 1



© 2010 SAP AG. All rights reserved. / Page 48

The default value of parameter **DeadlockDetectionLevel** is 4.

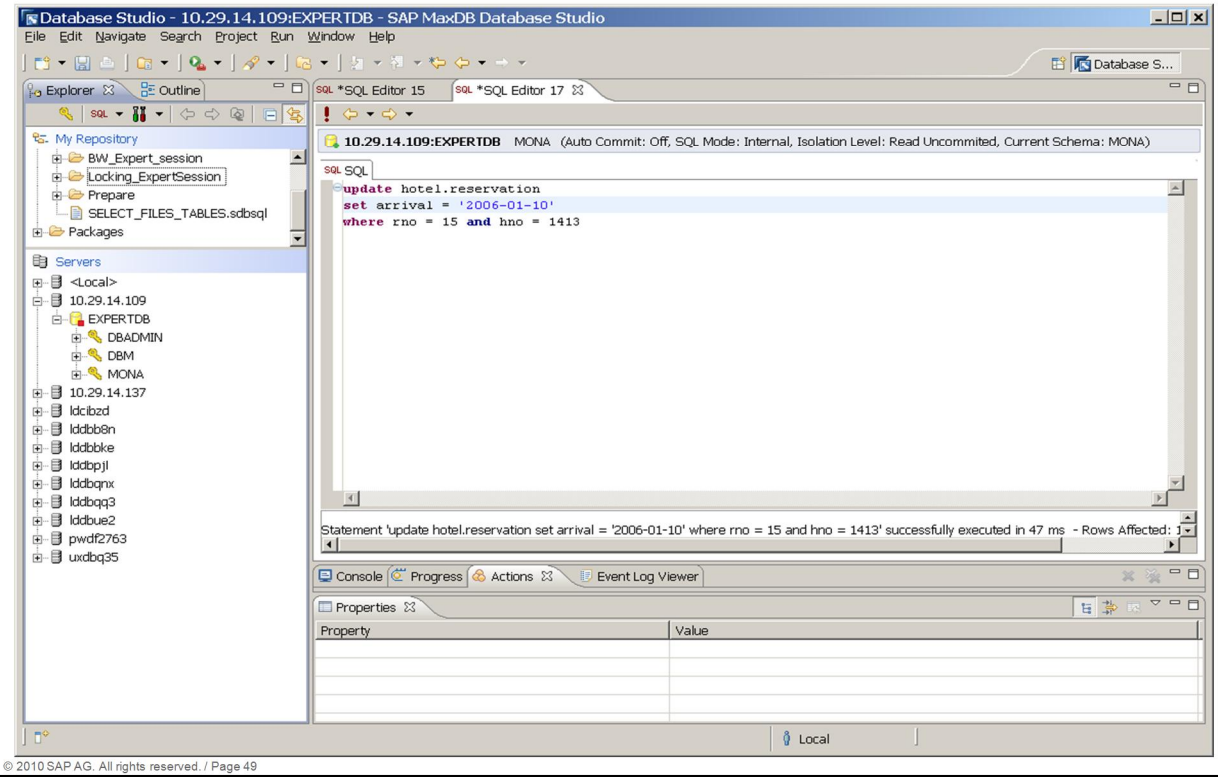
In the following example we have 2 transactions. Each transaction is holding a row exclusive lock on a table and is requesting an exclusive lock which is held by the other transaction.

First transaction is changing the price of single rooms for hotel 1413 in table *room*. This change is not committed yet.

Transaction 1 is holding a row exclusive lock.

6. Deadlock

6.2. Demo with DeadlockDetectionLevel = 4 : Transaction 2



© 2010 SAP AG. All rights reserved. / Page 49

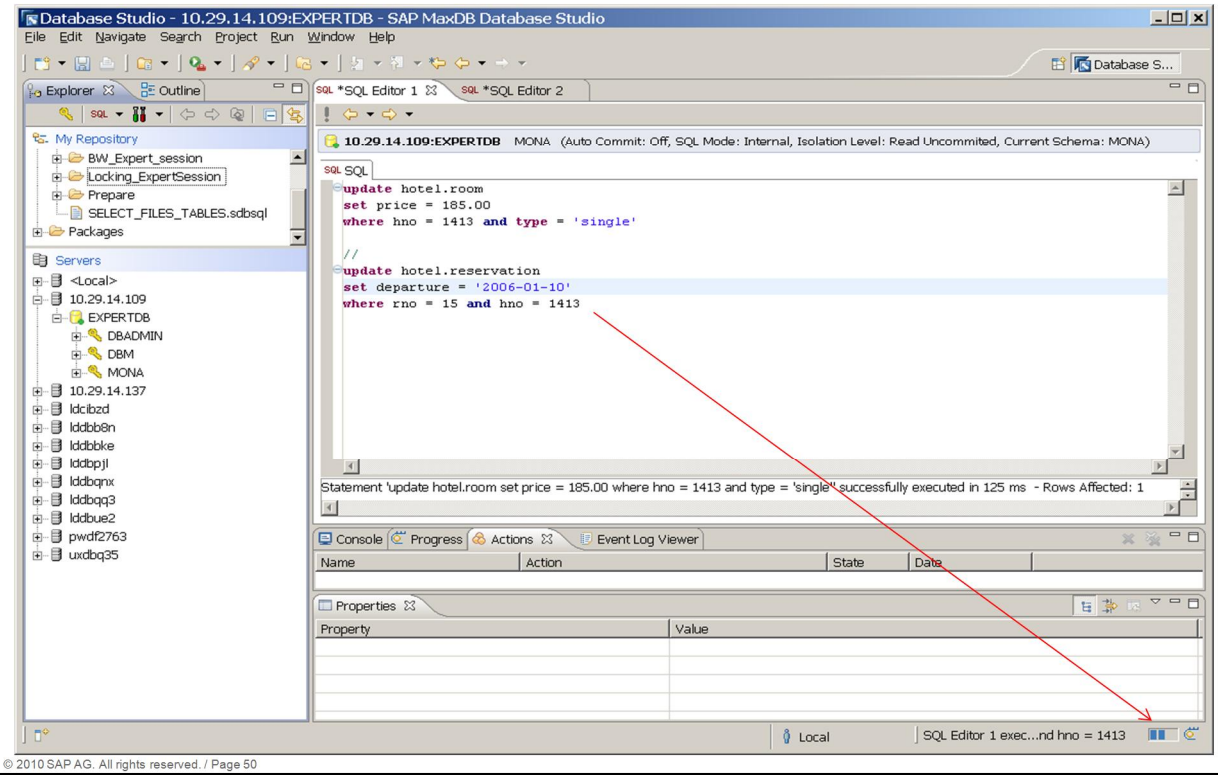
Second transaction is changing the arrival date for the room no 15 of hotel 1413 in table *reservation*.

Transaction 2 is holding a row lock exclusive on table reservation. This change is not committed yet.

At this time there is no wait or deadlock situation yet. We have 2 row exclusive on two different tables.

6. Deadlock

6.2. Demo: TA 1 is requesting the row lock of TA 2



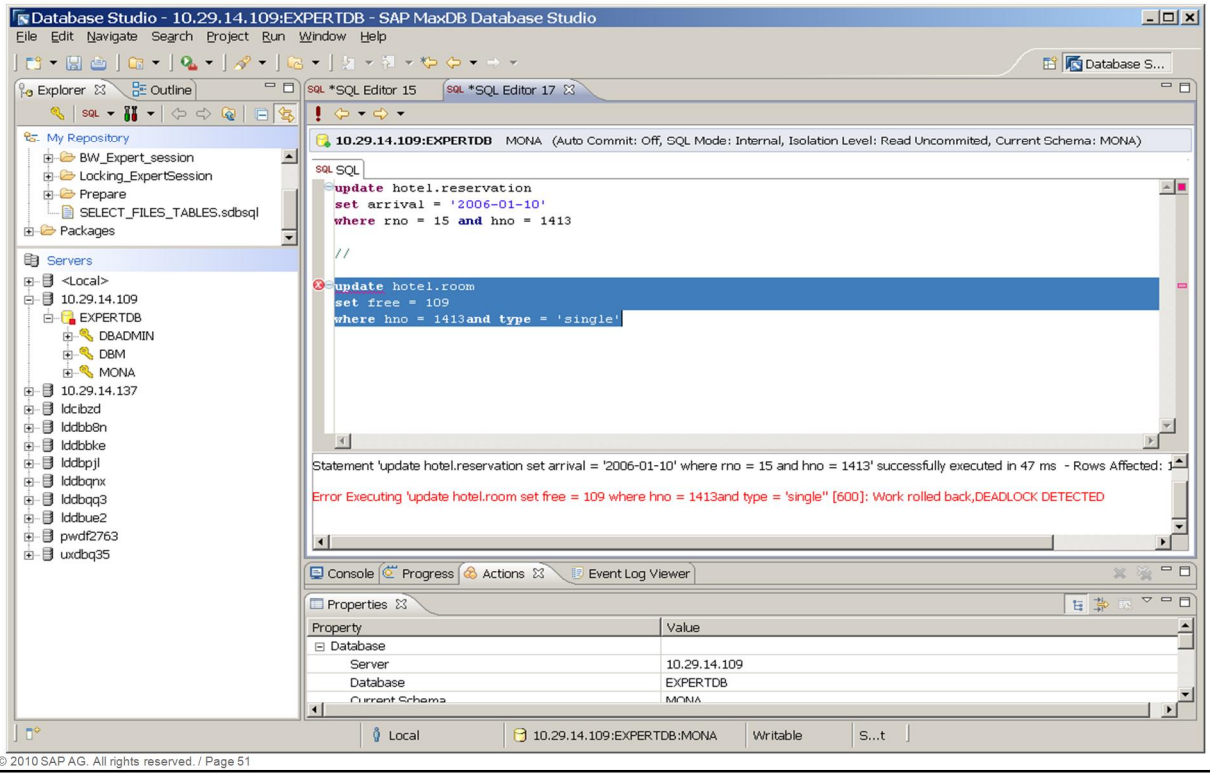
© 2010 SAP AG. All rights reserved. / Page 50

Transaction 1 is starting the second SQL command. Table *reservation* should be updated. The departure date of room no. 15 of hotel 1413 will be changed.

Because of transaction 2 is already holding the row lock exclusive on this row, transaction 1 has to wait. A request row lock is set.

6. Deadlock

6.2. Demo: Transaction 2: Deadlock detected



© 2010 SAP AG. All rights reserved. / Page 51

While transaction 1 is waiting for the row lock exclusive on table *reservation* which is hold by transaction 2, transaction 2 wants to do an update on table *room*.
The number of free rooms has to be reduced.
Because transaction 1 is holding a row exclusive on the same record of table *room* transaction 2 cannot get the row exclusive.

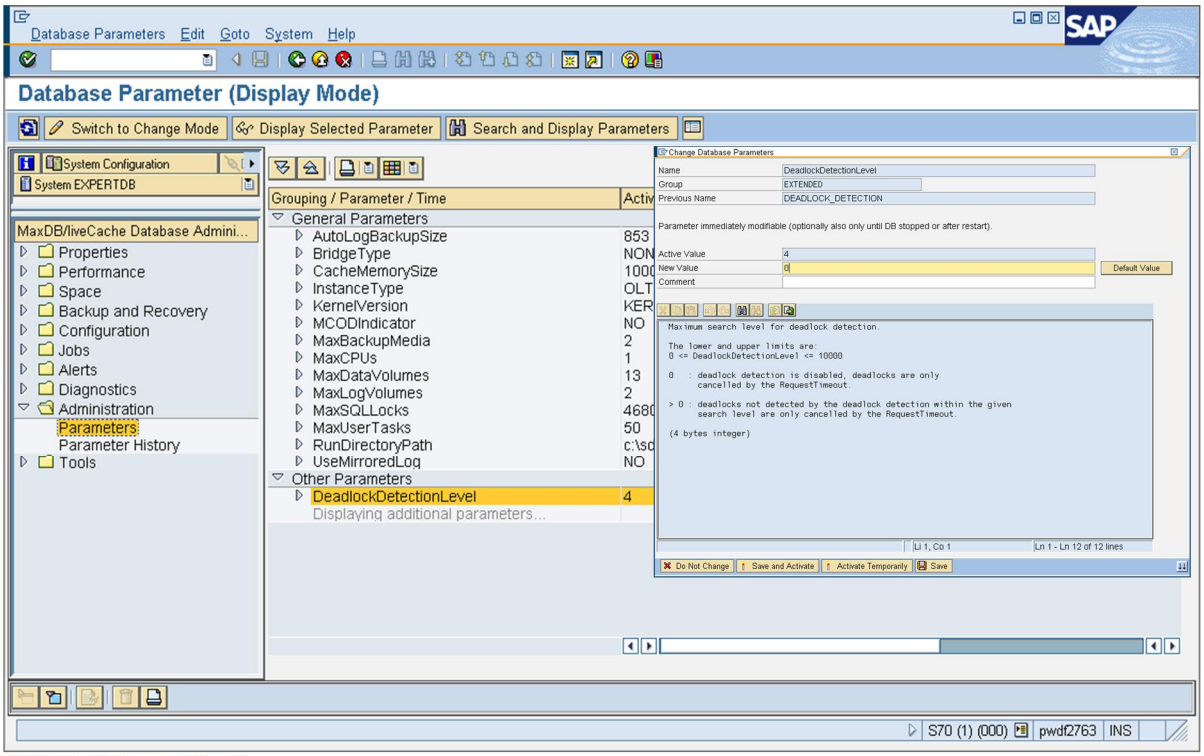
The deadlock detection mechanism will avoid this deadlock situation

- transaction 1 is holding the lock transaction 2 is waiting for
- transaction 2 is holding a lock transaction 1 is waiting for

with an implicit rollback of the second SQL command of transaction 2 (error message '-60 work rolled back, DEADLOCK DETECTED').

6. Deadlock

6.2. Demo: Parameter DeadlockDetectionLevel



© 2010 SAP AG. All rights reserved. / Page 52

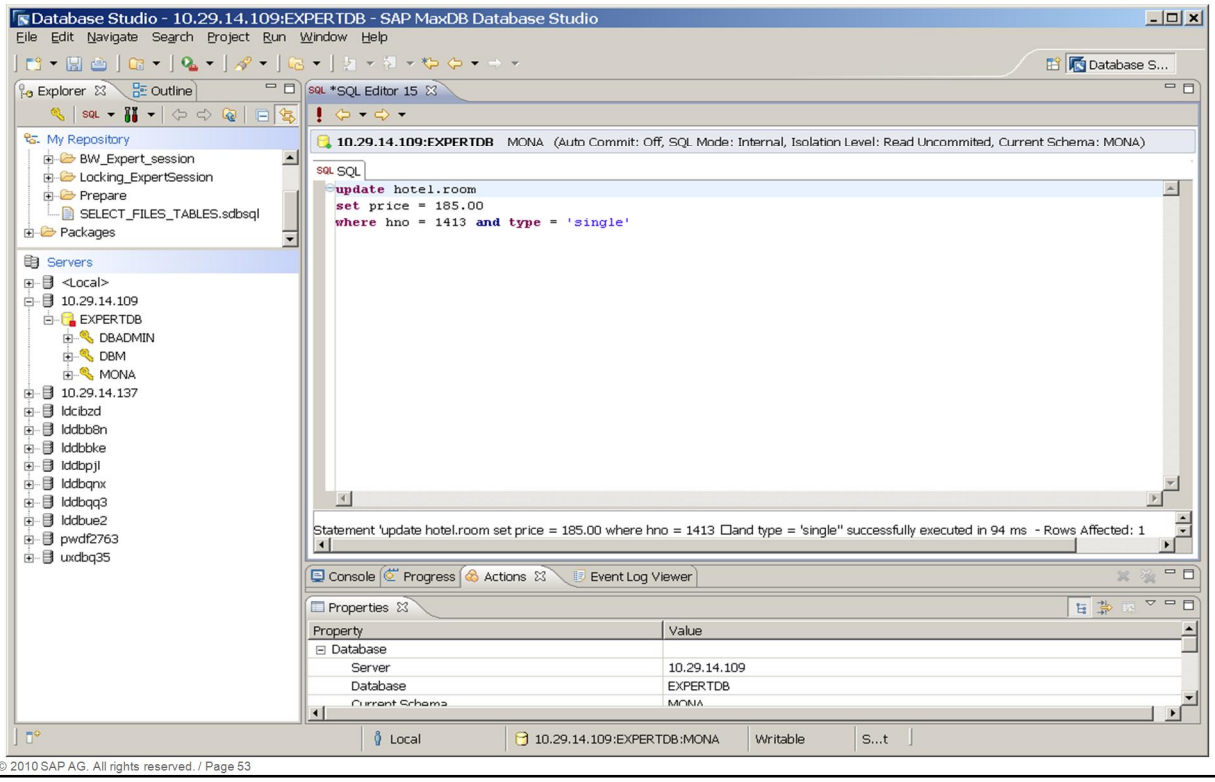
The database parameter ***DeadlockDetectionLevel*** defines the maximum depth (here: level 4) of search for the deadlock detection with SQL locks.

To check what happens during a deadlock the DeadlockDetectionLevel is switched off by setting the parameter to the value 0.

If the deadlock detection is switched off deadlocks will only implicitly be solved with request timeout mechanism.

6. Deadlock

6.3. Demo with DeadlockDetectionLevel = 0 : Transaction 1



© 2010 SAP AG. All rights reserved. / Page 53

The same example is starting again but deadlock detection is switched off now.

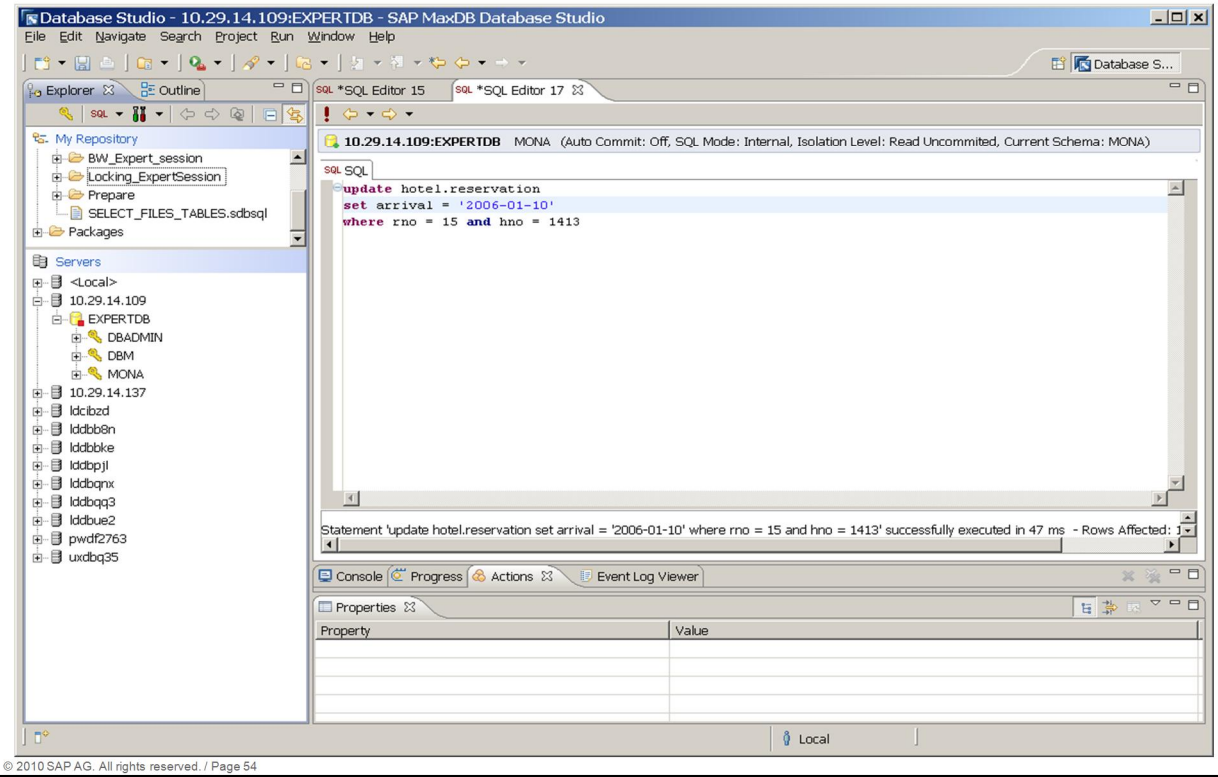
We have again 2 transactions. Each transaction is holding a row exclusive lock on a table and is requesting an exclusive lock which is held by the other transaction.

Transaction 1 is changing the price of single rooms for hotel 1413 in table *room*. This change is not committed yet.

Transaction 1 is holding a row exclusive lock.

6. Deadlock

6.3. Demo with DeadlockDetectionLevel = 0 : Transaction 2



© 2010 SAP AG. All rights reserved. / Page 54

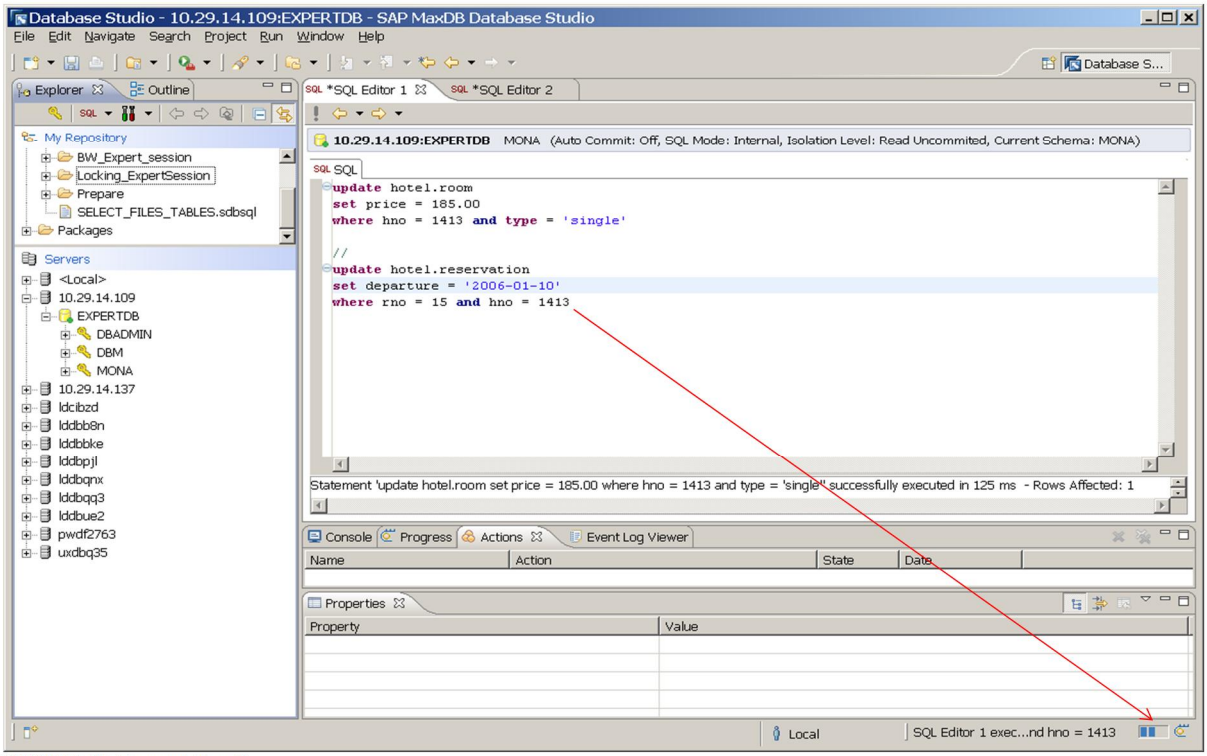
Transaction 2 is changing the arrival date for the room no 15 of hotel 1413 in table *reservation*.

Transaction 2 is holding a row lock exclusive on table *reservation*. This change is not committed yet.

There is no wait or deadlock situation yet. We have 2 row exclusive on two different tables.

6. Deadlock

6.3. Demo: TA 1 is waiting for Lock of TA 2



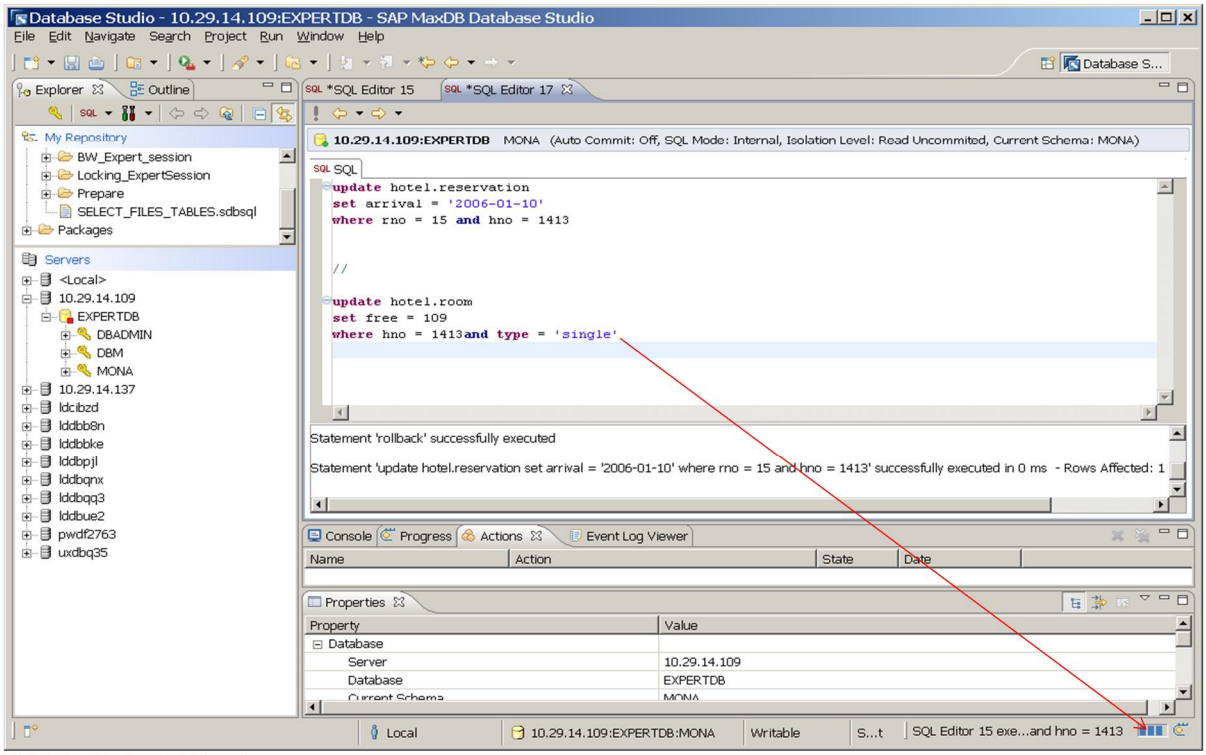
© 2010 SAP AG. All rights reserved. / Page 55

Transaction 1 is starting the second SQL command. Table *reservation* should be updated. The departure date of room no. 15 of hotel 1413 will be changed.

Because of transaction 2 is already holding the row lock exclusive on this row, transaction 1 has to wait. A request row lock is set.

6. Deadlock

6.3. Demo: TA 2 is waiting for Lock of TA 1



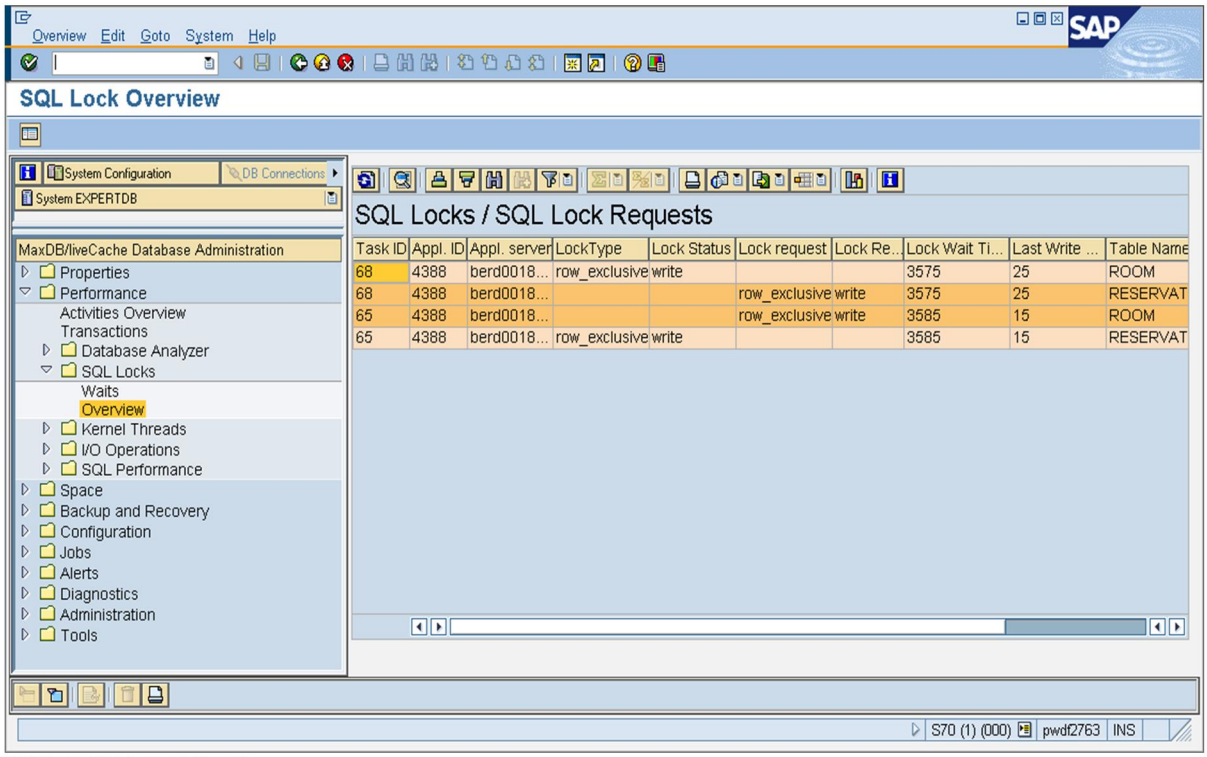
© 2010 SAP AG. All rights reserved. / Page 56

While transaction 1 is waiting for the row lock exclusive on table *reservation* which is hold by transaction 2, transaction 2 wants to do an update on table *room*. The number of free rooms has to be reduced.
Transaction 1 is holding a row exclusive on the same record of table *room* therefore transaction 2 cannot get the row exclusive.

Deadlock detection is switched off therefore both transactions are waiting for each other.
When the deadlock detection is switched off this deadlock situation will be solved when the request timeout is reached. One of those transactions is rolled back.

6. Deadlock

6.3. Demo: DBACOCKPIT Lock Overview



In transaction *DBACockpit -> Performance -> SQL Locks -> Overview* shows this current deadlock situation.

There exist 2 lock entries of Task ID 68:

- On table *room* an **lock row exclusive** (write lock).
- On table *reservation* a **request** for a **lock row exclusive** (write lock).

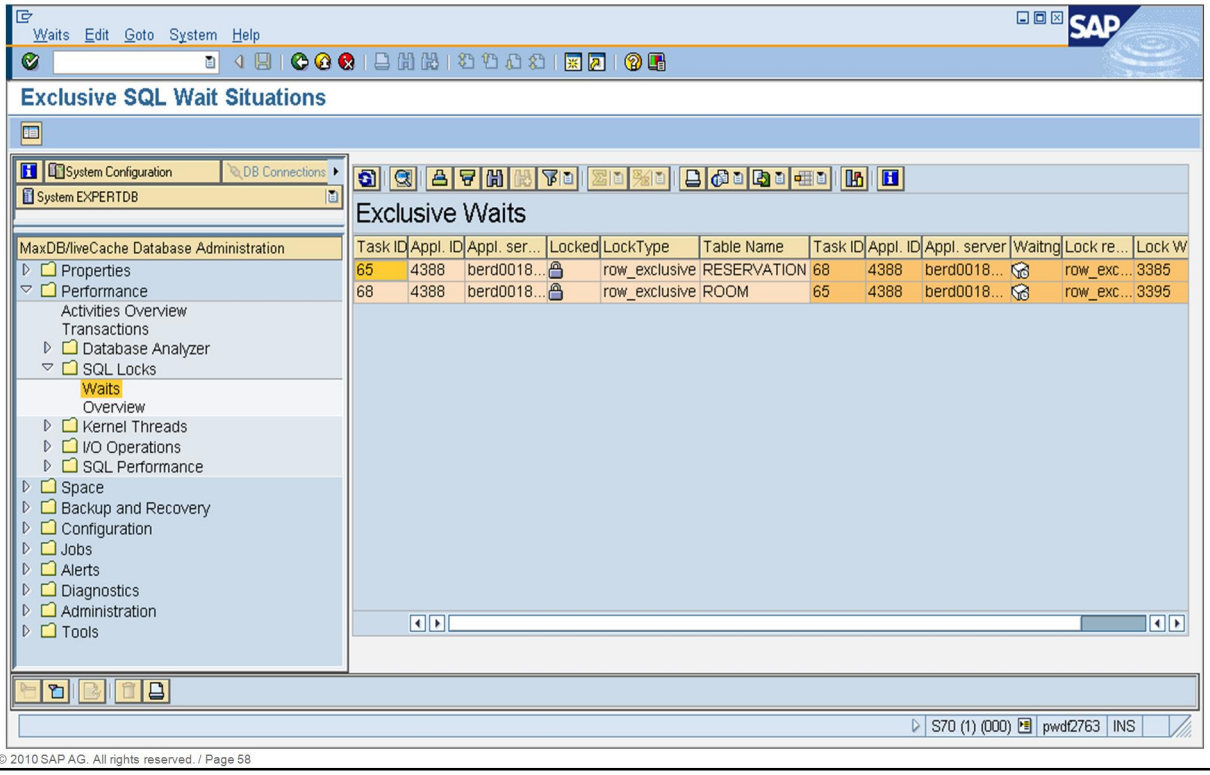
There exist 2 locks of Task ID 65:

- On table *reservation* a **lock row exclusive** (write lock).
- On table *room* a **request** for a **lock row exclusive** (write lock).

Notice: In MaxDB version ≤ 7.8 it is not possible to analyse deadlock situations in the past.

6. Deadlock

6.3. Demo: DBACOCKPIT Lock Waits



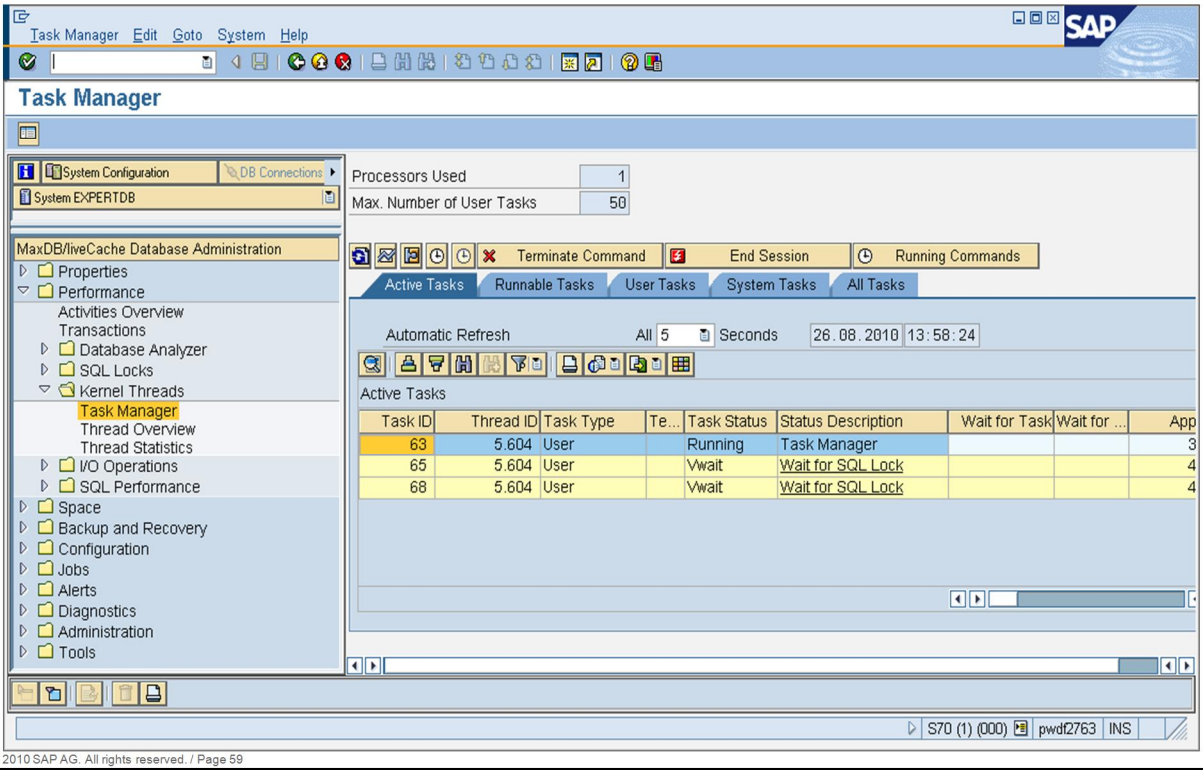
The deadlock situation can be better seen in transaction *DBACockpit -> Performance -> SQL Locks -> Waits*

Task ID 65 has an exclusive lock on table *reservation* which is requested from Task ID 68.

Task ID 68 has an exclusive lock on table *room* which is requested from Task ID 65.

6. Deadlock

6.3. Demo: DBACOCKPIT Task Manager

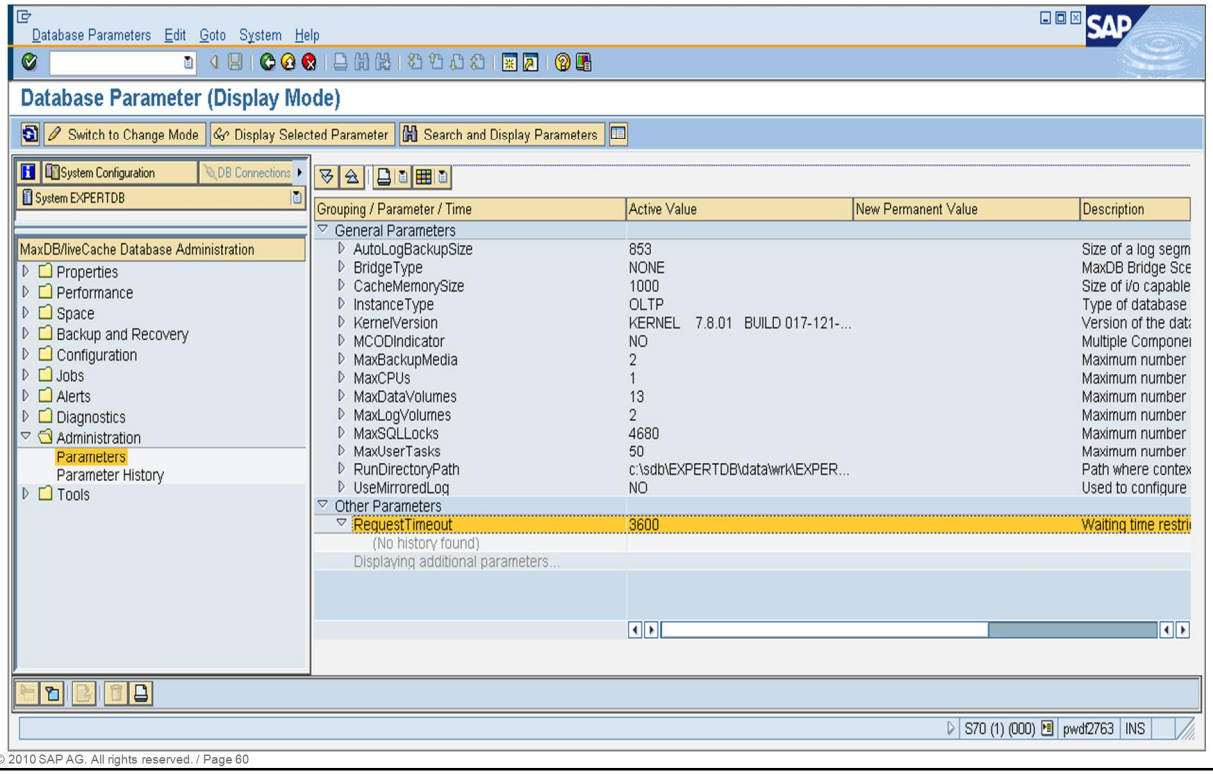


In the Task Manager of transaction *DBACockpit* the *Active Tasks* show that there are 2 tasks (65 & 86) in status *Vwait* which means that both transactions are waiting for SQL locks.

In this session's example we already know that these are those tasks which are currently in a deadlock situation.

6. Deadlock

6.3. Demo: Parameter RequestTimeout

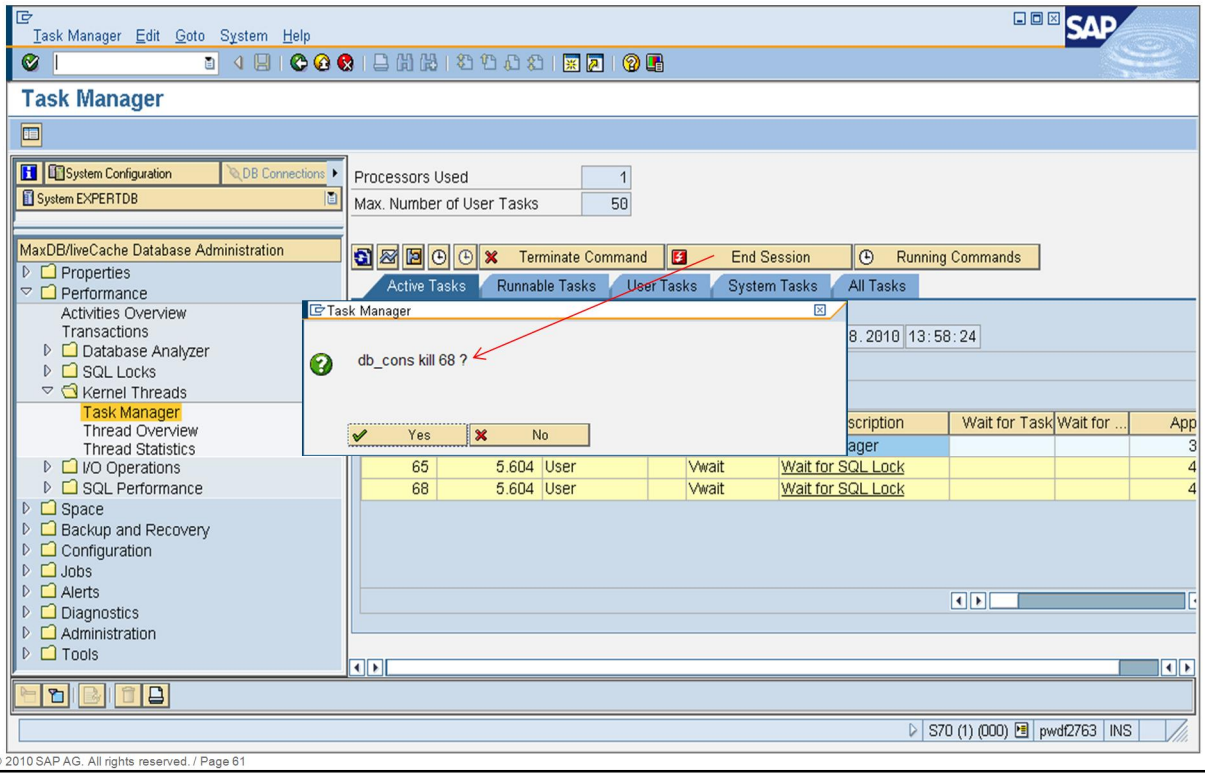


The deadlock situation can be solved implicitly by the request timeout, which is configured with parameter **RequestTimeout.** In SAP environment the parameter RequestTimeout is set to 3600 seconds.
When the wait time of request timeout is reached the transaction which is requesting the lock will be rolled back with error message '*-51 Lock request timeout*'.

For this session we don't want to wait 1 hour until the deadlock will be solved implicitly. The *Task Manager* can be used to kill one of the in the deadlock involved tasks (=transactions).

6. Deadlock

6.3. Demo: DBACOCKPIT Task Manager: End Session

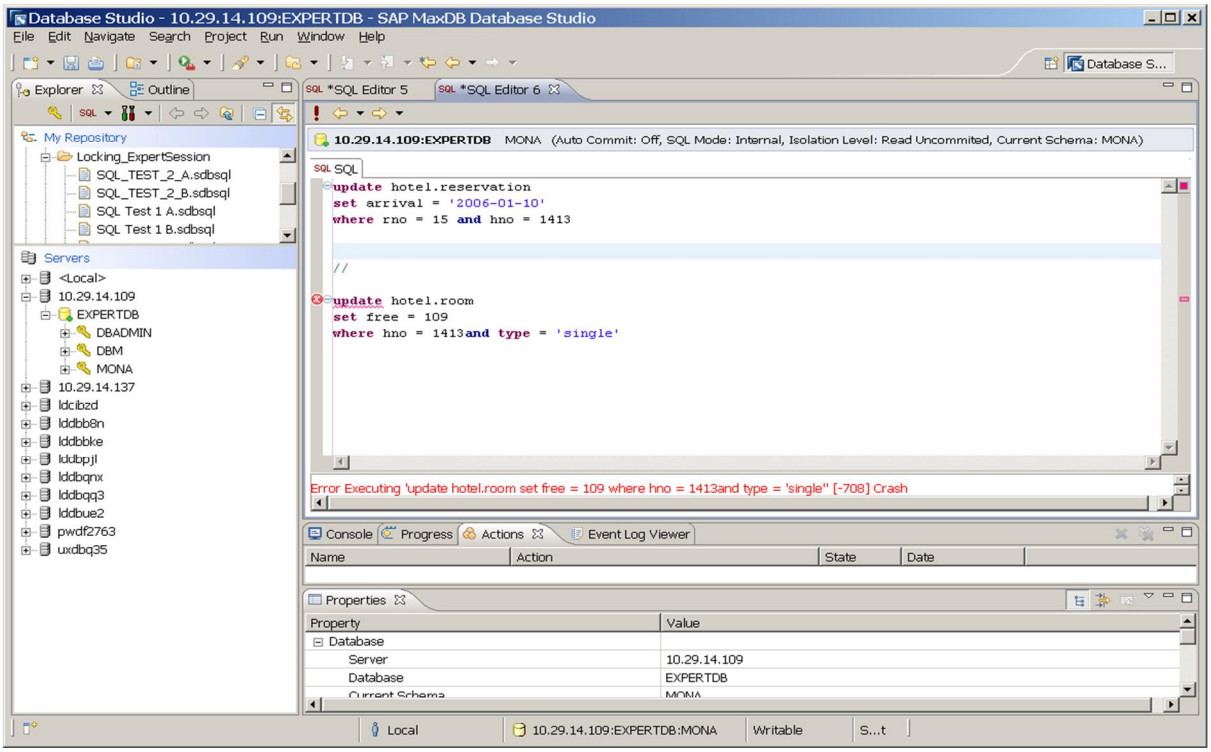


To end a session in the *Task Manager* the task ID has to be marked and with *End Session* the task will be killed.

With *Terminate Command* only the SQL command is canceled not the complete database session.

6. Deadlock

6.3. Demo: Error Message for TA 2

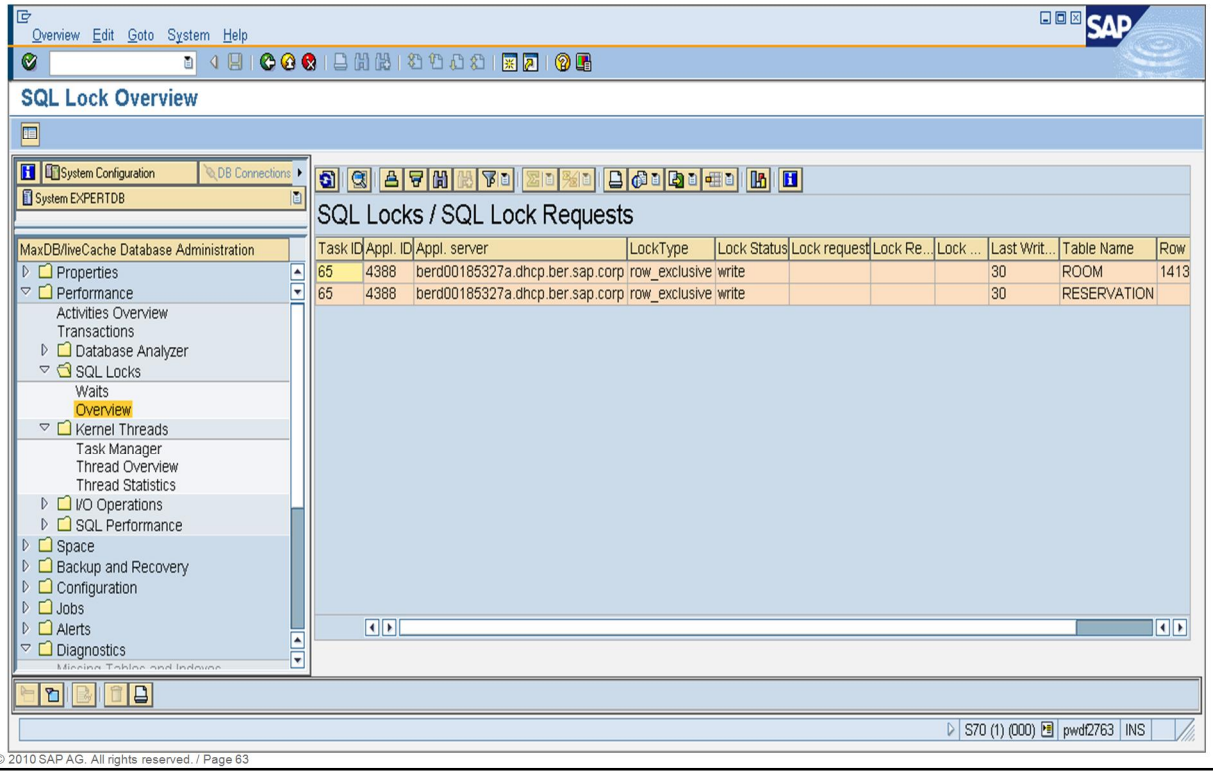


© 2010 SAP AG. All rights reserved. / Page 62

After the task has been killed via *Task Manager* in transaction *DBACockpit* you'll get an error message in Database Studio for transaction 2.

6. Deadlock

6.3. Demo: DBACOCKPIT Lock Overview

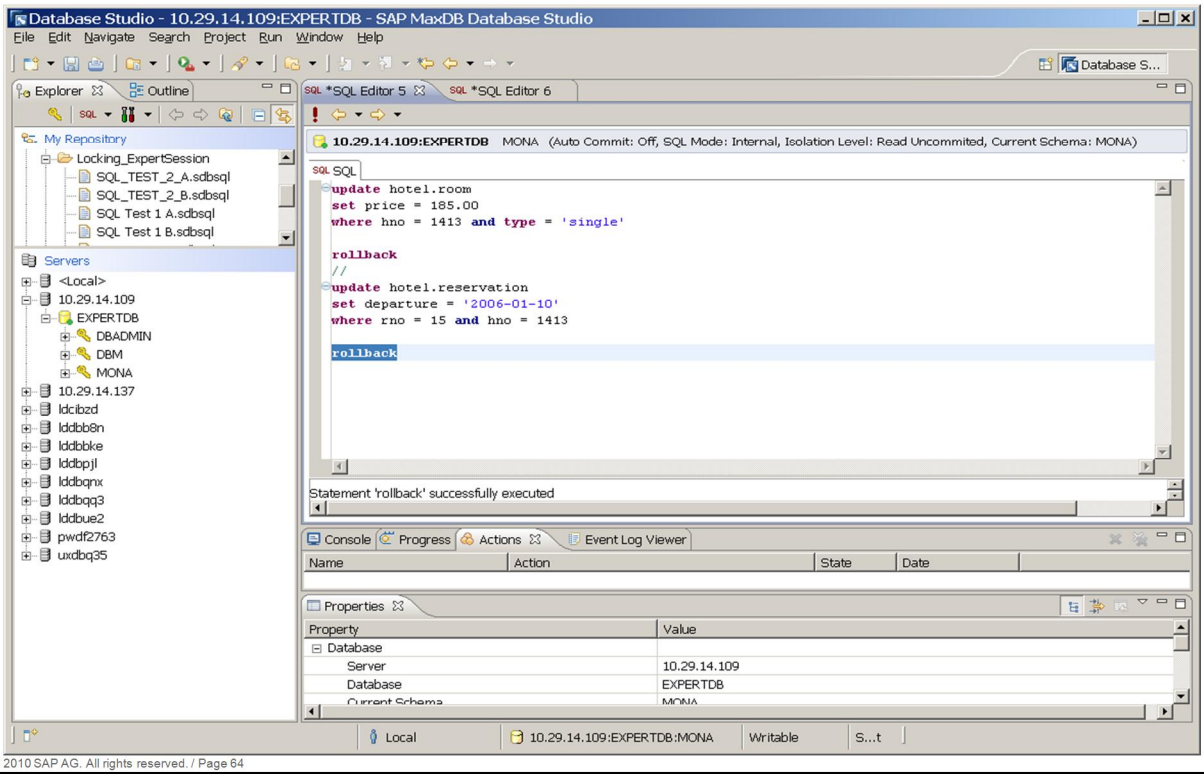


In transaction *DBACockpit -> Performance -> SQL Locks -> Overview* task ID 68 which was holding a lock on table *room* and requesting lock on table *reservation* disappeared.

Task ID 65 (Transaction 1) still holds the lock on table *reservation* and got the lock on table *room* as well.

6. Deadlock

6.3. Demo: Transaction 1: Rollback

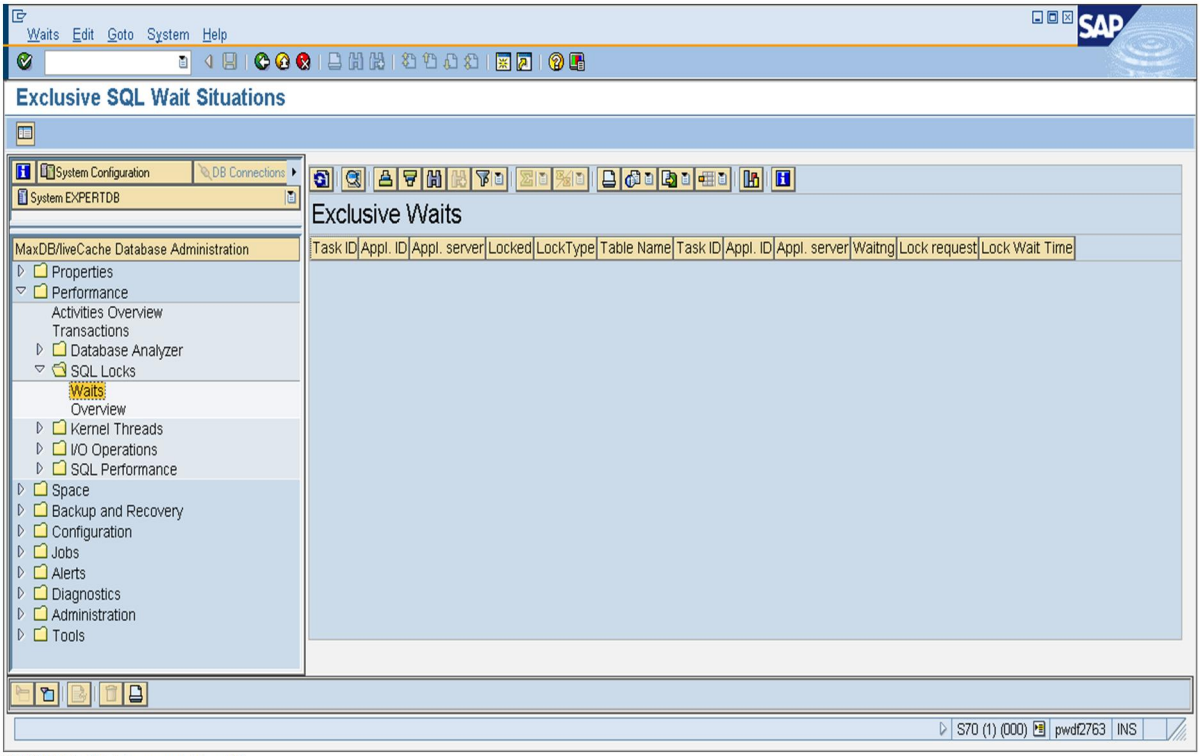


© 2010 SAP AG. All rights reserved. / Page 64

As soon as transaction T1 is finished here with an explicit rollback all locks are released.

6. Deadlock

6.3. Demo: DBACOCKPIT Lock Waits



© 2010 SAP AG. All rights reserved. / Page 65

In transaction *DBACockpit -> Performance -> SQL Locks -> Waits* it can be seen that no lock wait situation exists anymore.

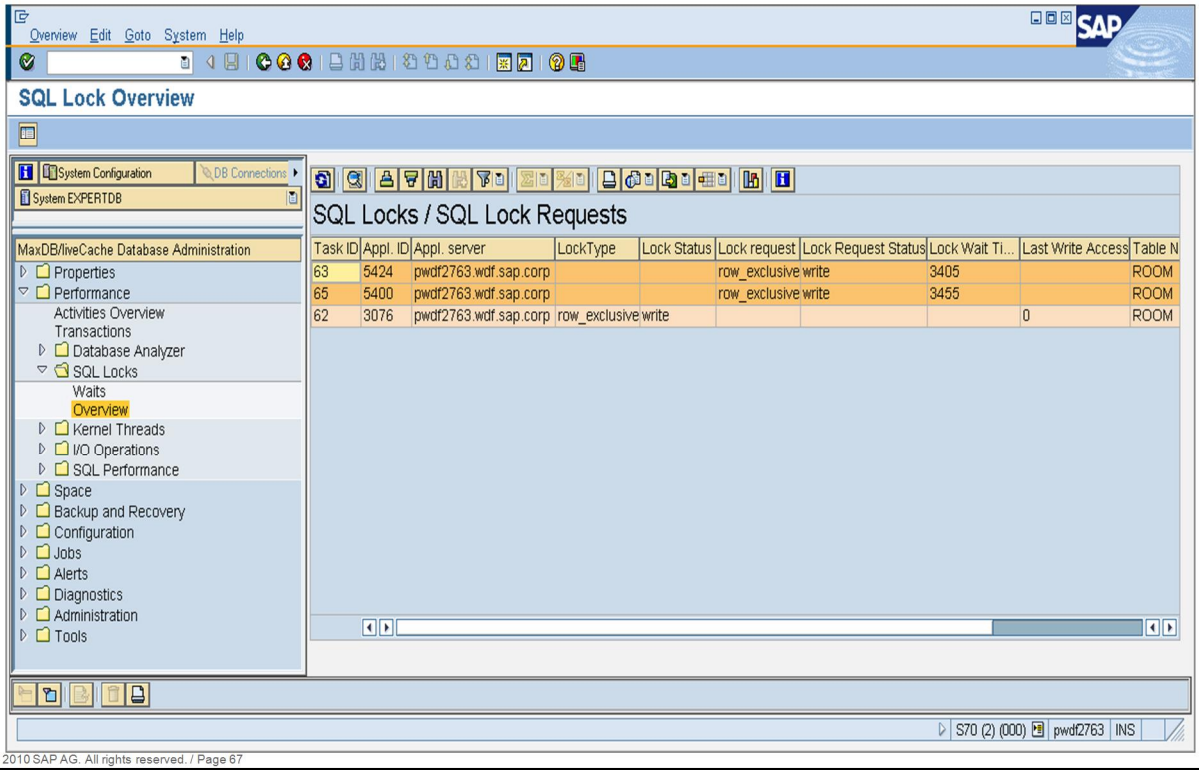
Agenda

1. Introduction
2. Lock Objects And Locking Types
3. Phenomena And Isolation Level
4. System Monitoring
5. Lock Escalation
6. Deadlock
7. Which Application Blocks The System?
8. Summary

All rights reserved.

7. Which Application Blocks The System?

7.1. Demo: DBACOCKPIT Locks Overview



© 2010 SAP AG. All rights reserved. / Page 67

Another important issue to know is how to analyze system blocking situations. To solve a blocking situation you need to know which application is responsible for the lock which blocks the system.

In transaction *DBACockpit -> Performance -> SQL Locks –> Overview* we have 3 transactions (Task ID: 63, 65, and 62).

All 3 transactions are started on the same application server. The application ID (*Appl. ID*) is important to know for further analysis.
The application ID is the application process ID (application PID) of the work process which is connected to the database task.

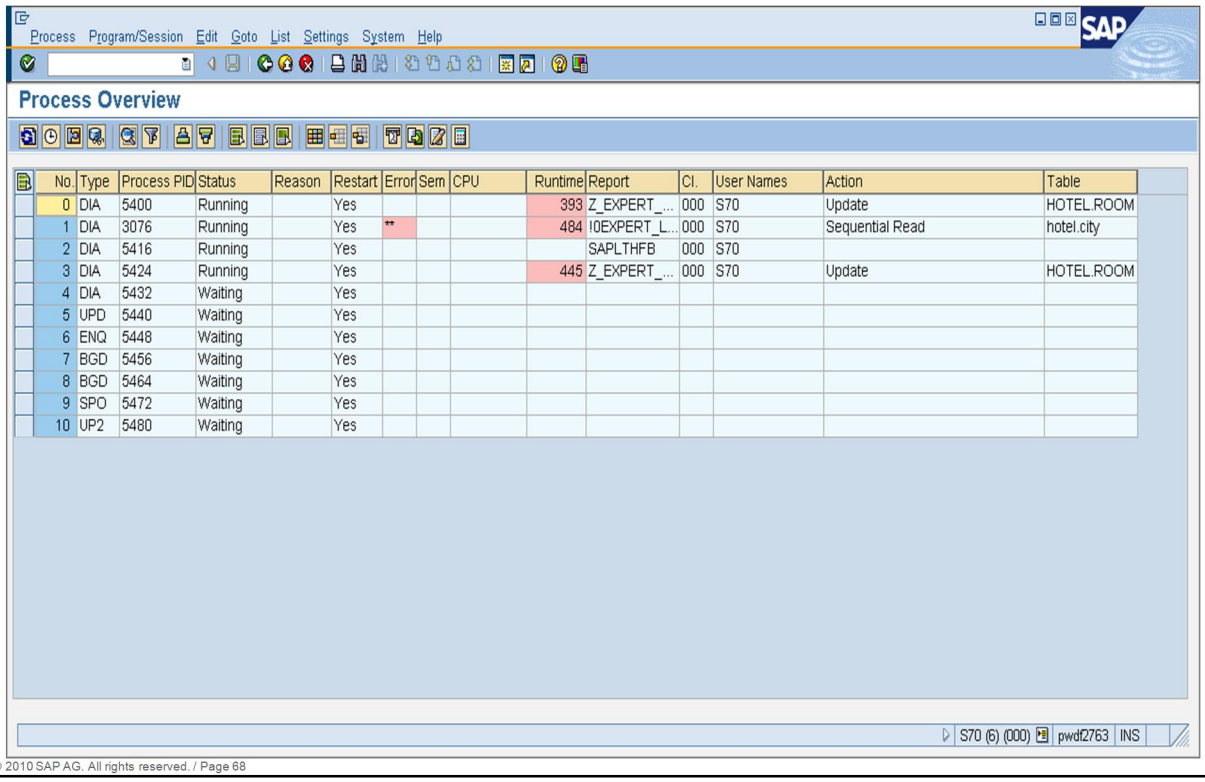
Workprocess:	Database Task:
5424	63
5400	65
3076	62

Application 3076 has an exclusive lock on table *room.*

Application 5400 and 5424 are requesting an exclusive lock on table *room*.

7. Which Application Blocks The System?

7.1. Demo: SM50 Process Overview



© 2010 SAP AG. All rights reserved. / Page 68

All applications which are involved in this hanging situation are running on the same application server. To get an overview transaction SM50 (Process Overview) on the application server pwdf2763 is used to get more detailed information.

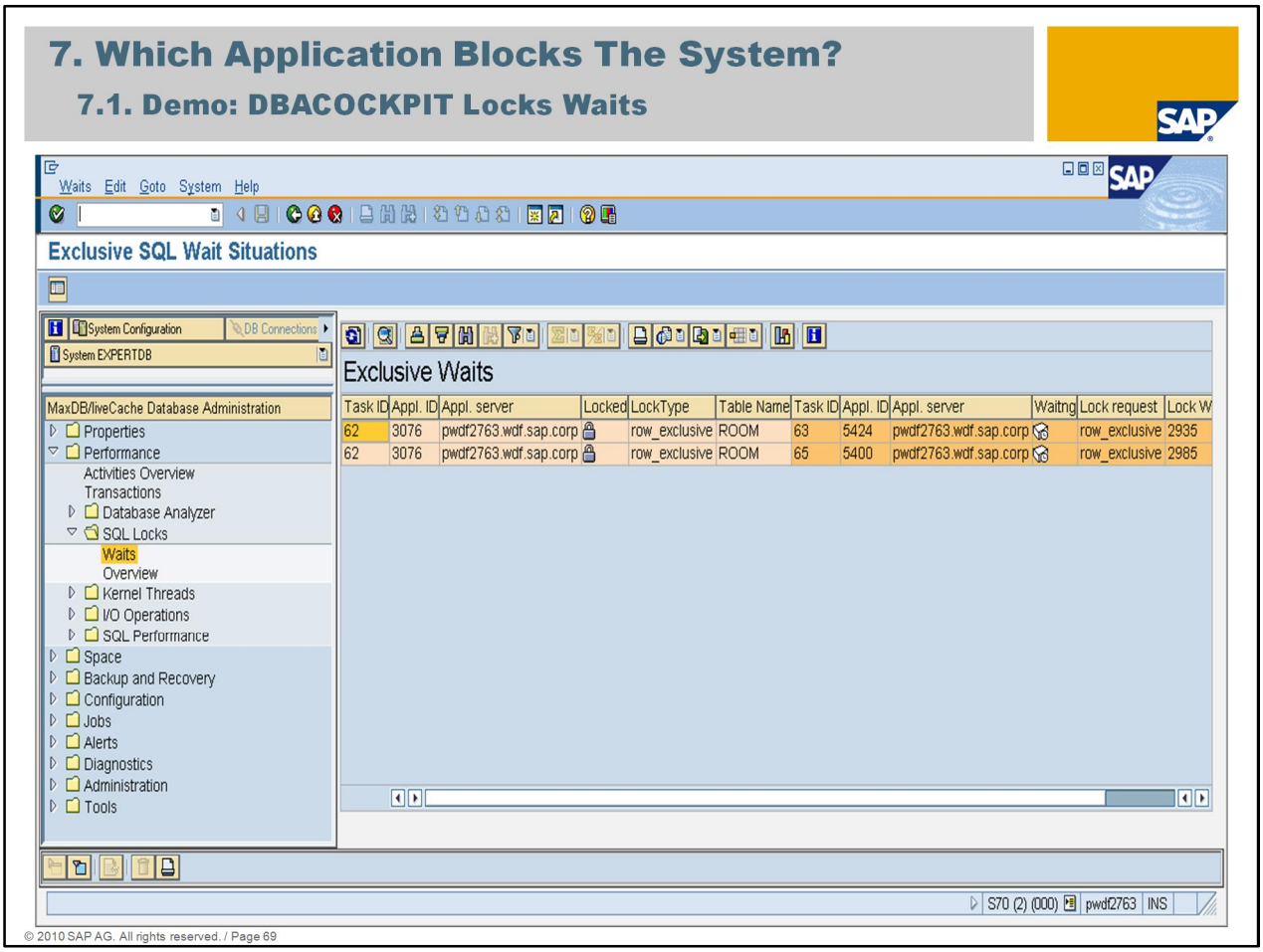
If applications on several servers are involved into a hanging situation transaction SM66 (Global Work Process Overview) is used for detailed analysis.

Work process 0 has application PID 5400, which is in status *Running*. Currently in action Update on table *room*.

Work process 1 has application PID 3076, which is in status *Running* as well. Currently doing a sequential read on table *city*.

Work process 2 is not relevant for the blocking situation.

Work process 3 has application PID 5424, which is in status *Running* as well. Currently doing an Update on table *room*.



To get the information which transaction is waiting for which lock transaction *DBACockpit -> Performance -> SQL Locks -> Waits* is used.

Application PID 3067 (Work process 1) is holding the exclusive lock and both Task ID 63 (WP 3) and Task ID 65 (WP 0) are waiting for this lock.

From the output of transaction SM50 task ID 62 (WP 1) is doing a sequential read on table *city* and from database point of view even has a lock on table *room*.

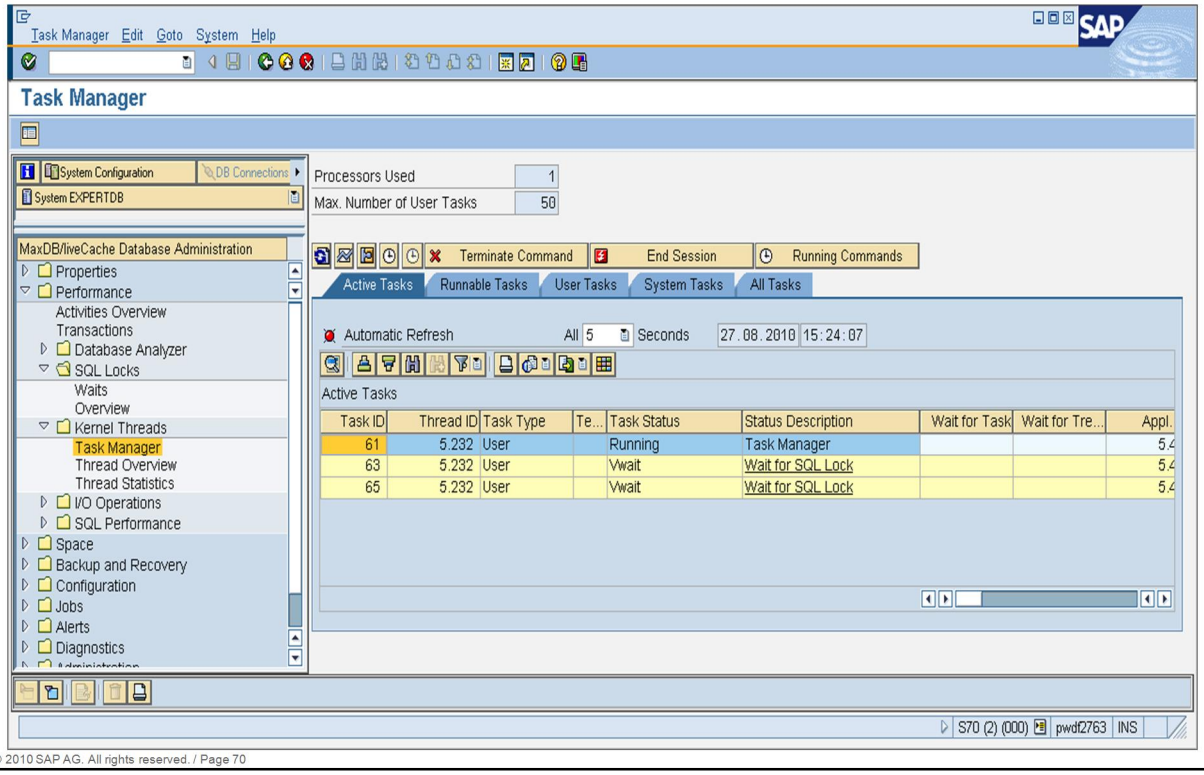
This looks like in the ABAP program after an update, insert or delete there is no commit work.

This may not happen in a regular SAP environment where the application programmers are using the SAP Enqueue mechanism to lock and unlock table accesses.

This can only happen when you are using native SQL in your ABAP coding, which is not recommended.

7. Which Application Blocks The System?

7.1. Demo: DBACOCKPIT Task Manager (1)



© 2010 SAP AG. All rights reserved. / Page 70

Additionally transaction *DBACockpit -> Performance -> Kernel Threads -> Task Manager -> Active Tasks* is used to check the status of each involved tasks from database side of view.

Only those tasks are listed in *Active Tasks* which are active on the database level.

On database level Task ID 63 and 65 are in status *Vwait*, which means they are waiting for SQL locks.

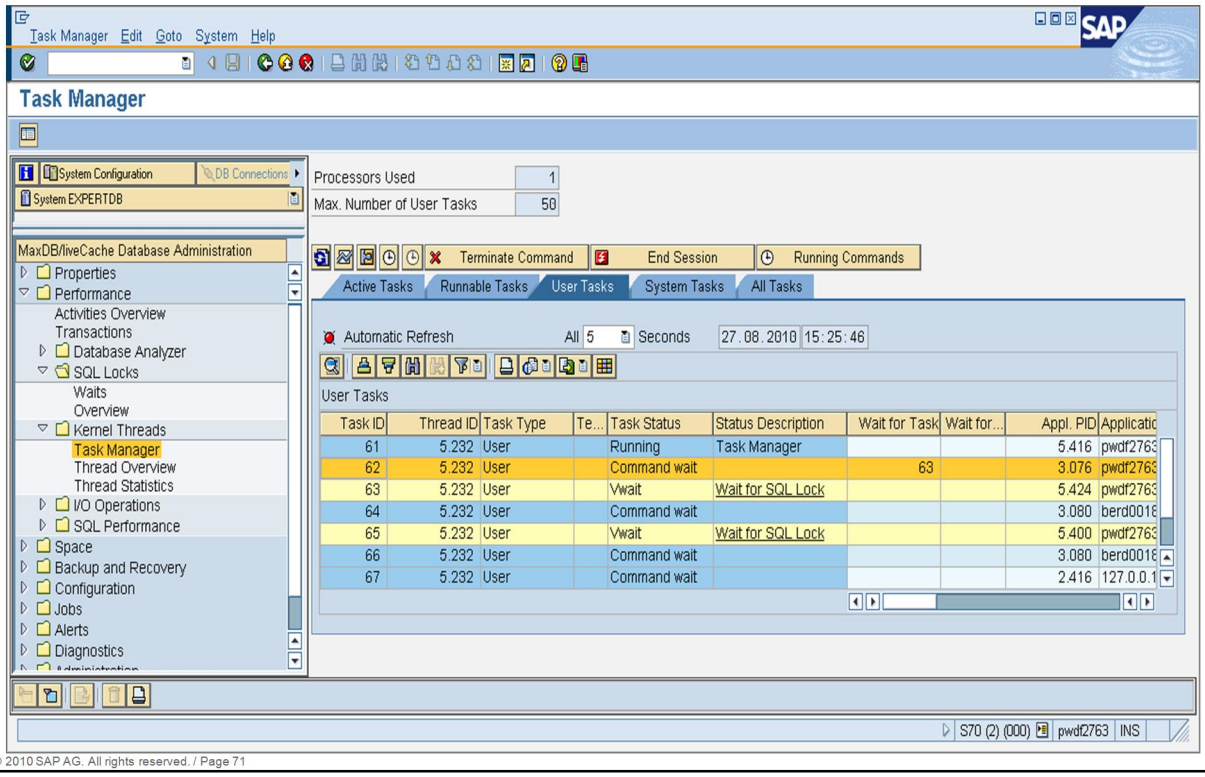
Task ID 61 is the Task Manager itself, so this is not relevant for the analysis of the blocking situation.

The third task ID 62, which is involved into the blocking situation is missing in the *Active Tasks.* Why?

It looks like this task is doing nothing in the database kernel.

7. Which Application Blocks The System?

7.1. Demo: DBACOCKPIT Task Manager (2)



© 2010 SAP AG. All rights reserved. / Page 71

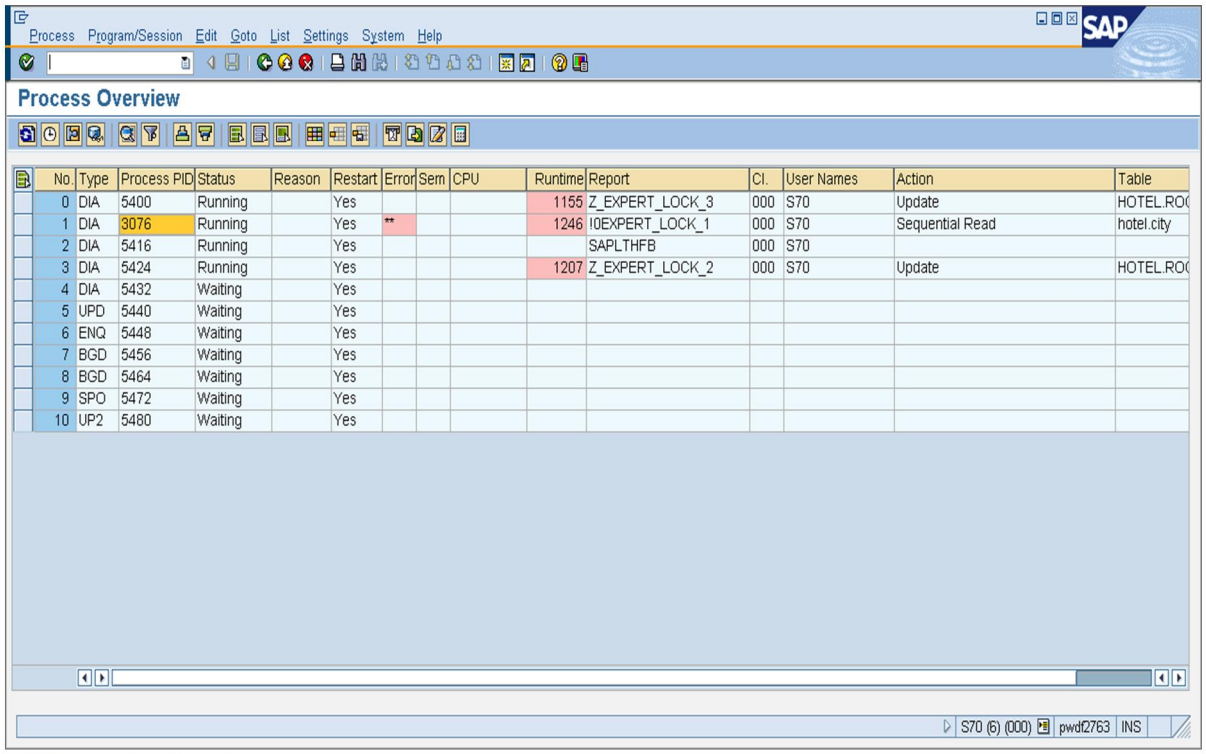
An overview about the status of all user tasks can be seen with *User Tasks*.
Task ID 62 is in status *Command wait.* Therefore there is no entry in *Active Tasks.*

A Task which is in status *Command wait* is doing nothing on the database. The task is waiting for the next application command which is sent to the database.

Conclusion this task must be terminated to get the blocking situation solved.

7. Which Application Blocks The System?

7.1. Demo: SM50 Process Overview



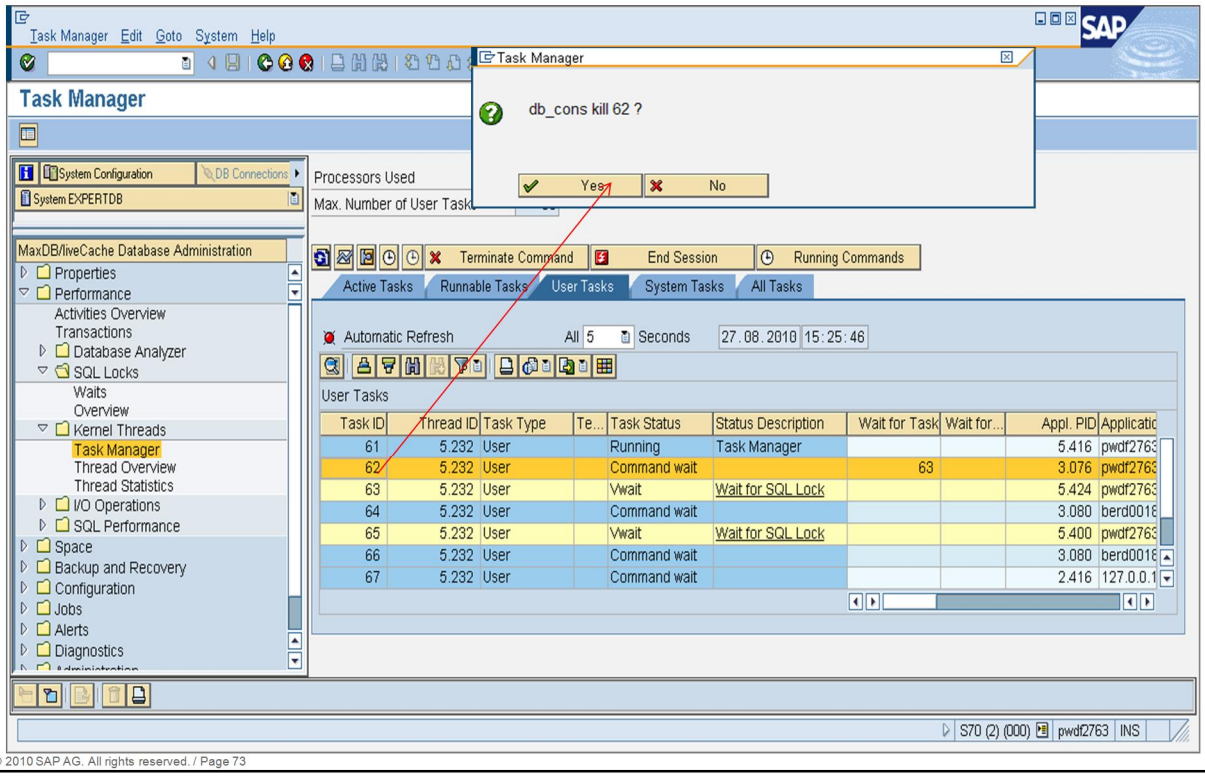
© 2010 SAP AG. All rights reserved. / Page 72

Notice: Even when you see in transaction SM50 that application 3076 (Task ID 62) is busy with sequential read on a table, this does not mean that the application is busy on the database.

Therefore always use the MaxDB *DBACockpit (or DB50) -> Task Manager* to analyze and solve blocking situations.

7. Which Application Blocks The System?

7.1. Demo: DBACOCKPIT Task Manager

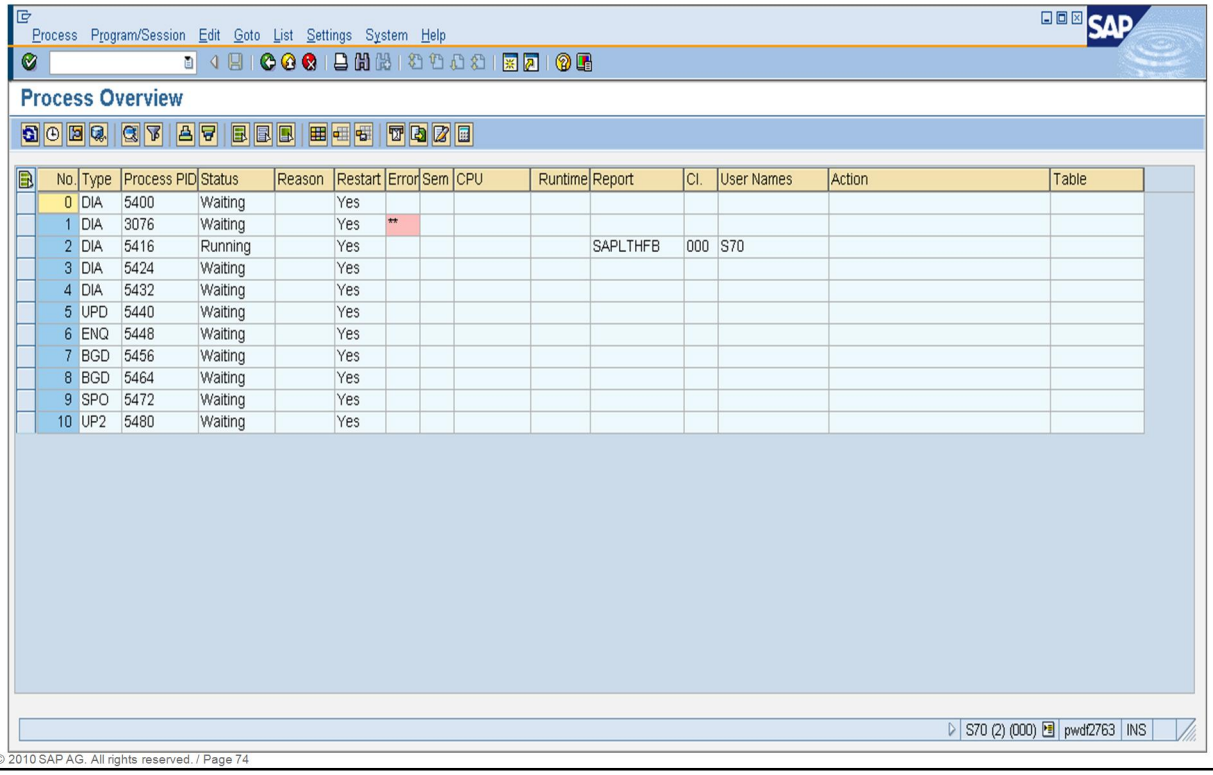


© 2010 SAP AG. All rights reserved. / Page 73

With transaction *DBACockpit -> Performance -> Kernel Threads -> Task Manager* Task ID 62 will be killed by using *End Session*.

7. Which Application Blocks The System?

7.1. Demo: SM50 Process Overview



© 2010 SAP AG. All rights reserved. / Page 74

After the Task ID 62 (application 3076) has been killed via MaxDB *Task Manager* the blocking situation has been solved.

Agenda

1. Introduction
2. Lock Objects And Locking Types
3. Phenomena And Isolation Level
4. System Monitoring
5. Lock Escalation
6. Deadlock
7. Which Application Blocks The System?
8. Summary

All rights reserved.

8. Summary

8.1. Configuring lock management

DB Kernel Parameters

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

© 2010 SAP AG. All rights reserved. / Page 76

8. Summary

8.2. Error messages

DB Error Messages regarding Lock Management

‘-1000 Too many lock requests’	too many SQL locks were requested, maximum value (MaxSQLLocks) has been reached
‘-51 Lock request timeout’	maximum wait time for the lock assignment (RequestTimeout) has expired
‘-60 Work rolled back’	deadlock detected

© 2010 SAP AG. All rights reserved. / Page 77

8. Summary

8.3. Information resources

- SAP notes:

  1243937 FAQ: MaxDB SQL Locks

  65946 Lock Escalations

- MaxDB documentation:

  http://maxdb.sap.com

  → Documentation

  → Version 7.8

  → Glossary (‚Lock' or ‚Isolation Level')

- MaxDB classroom training:

  WB550 Workshop MaxDB Internals

© 2010 SAP AG. All rights reserved. / Page 78

Questions and Answers

Thank You!
Bye, Bye – And Stay Tuned for Further Sessions

	Feedback and further information: http://www.sdn.sap.com/irj/sdn/maxdb
	Next expert sessions are:
	External Backup Tools – 2010-10-05
	Tracing: SQLDBC, ODBC & VTRACE usage scenarios – 2010-11-09

© 2010 SAP AG. All rights reserved. / Page 80

Thanks a lot for your interest in this session.

The recording of this session will be available on maxdb.sap.com/training soon. There you'll find also the recordings of all previously held Expert Sessions about SAP MaxDB.

Further Expert Sessions about SAP MaxDB are currently planned. It will be posted on our SDN page http://www.sdn.sap.com/irj/sdn/maxdb if and when these sessions will take place.

© 2010 SAP AG. All Rights Reserved

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft, Windows, Excel, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.

IBM, DB2, DB2 Universal Database, System i, System i5, System p, System p5, System x, System z, System z10, System z9, z10, z9, iSeries, pSeries, xSeries, zSeries, eServer, z/VM, z/OS, i5/OS, S/390, OS/390, OS/400, AS/400, S/390 Parallel Enterprise Server, PowerVM, Power Architecture, POWER6+, POWER6, POWER5+, POWER5, POWER, OpenPower, PowerPC, BatchPipes, BladeCenter, System Storage, GPFS, HACMP, RETAIN, DB2 Connect, RACF, Redbooks, OS/2, Parallel Sysplex, MVS/ESA, AIX, Intelligent Miner, WebSphere, Netfinity, Tivoli and Informix are trademarks or registered trademarks of IBM Corporation.

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

Adobe, the Adobe logo, Acrobat, PostScript, and Reader are either trademarks or registered trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Oracle is a registered trademark of Oracle Corporation.

UNIX, X/Open, OSF/1, and Motif are registered trademarks of the Open Group.

Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.

HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

Java is a registered trademark of Sun Microsystems, Inc.

JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

SAP, R/3, SAP NetWeaver, Duet, PartnerEdge, ByDesign, Clear Enterprise, SAP BusinessObjects Explorer and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and other countries.

Business Objects and the Business Objects logo, BusinessObjects, Crystal Reports, Crystal Decisions, Web Intelligence, Xcelsius, and other Business Objects products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP France in the United States and in other countries.

All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.

The information in this document is proprietary to SAP. No part of this document may be reproduced, copied, or transmitted in any form or for any purpose without the express prior written permission of SAP AG.

This document is a preliminary version and not subject to your license agreement or any other agreement with SAP. This document contains only intended strategies, developments, and functionalities of the SAP® product and is not intended to be binding upon SAP to any particular course of business, product strategy, and/or development. Please note that this document is subject to change and may be changed by SAP at any time without notice.

SAP assumes no responsibility for errors or omissions in this document. SAP does not warrant the accuracy or completeness of the information, text, graphics, links, or other items contained within this material. This document is provided without a warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, or non-infringement.

SAP shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials. This limitation shall not apply in cases of intent or gross negligence.

The statutory liability for personal injury and defective products is not affected. SAP has no control over the information that you may access through the use of hot links contained in these materials and does not endorse your use of third-party Web pages nor provide any warranty whatsoever relating to third-party Web pages.

© 2010 SAP AG. All rights reserved. / Page 81