

Analysing and Resolving Deadlocks

Erland Sommarskog

SPONSORS:

NTT DATA
FOUR9S
V-Valley
Coyote Monitor by Aleson ITC
encamina PIENSA EN COLORES
Next Step
prodware
TD SYNNEX
ALTIA
waibbo



Erland Sommarskog

Independent consultant based in Stockholm

SQL Server MVP since 2001

esquel@sommarskog.se

http://www.sommarskog.se

Slides and scripts are available on
http://www.sommarskog.se/present

© Erland Sommarskog 2026

PLATINO

NTT DATA

FOUR9S

V-Valley

gold

Coyote Monitor
by Aleson ITC

encamina
PIENSA EN COLORES

Next Step

prodware

TD SYNNEX

silver

ALTIA

waibbo

Agenda

- Introduction to deadlocks.
- How to get information about deadlocks?
- Understanding the deadlock XML.
- Quick recap on locking.
- Isolation levels and deadlock prevention.
- More means to prevent deadlocks.
- Retry on deadlock.

© Erland Sommarskog 2026

What Is a Deadlock?

01_FirstWindow.sql
01_SecondWindow.sql

- A deadlock is when two or more processes are blocking each other in such a way that neither can continue.
- SQL Server checks for this condition every five seconds.
- If a deadlock is found, SQL Server injects an error in one of the processes, the *deadlock victim*, and rolls back its transaction.
  - The deadlock victim is selected based on the amount of log records.

© Erland Sommarskog 2026

Are Deadlocks a Problem?

- Deadlocks occur in the best of families.
- An occasional deadlock is no cause for alarm.
- But many deadlocks per day often are.
- Users don't like errors about being "deadlock victims".
- Too many deadlocks can reduce throughput, since it takes a few seconds until a deadlock is resolved.

© Erland Sommarskog 2026

What Resources Can Cause Deadlock

- Message 1205 in sys.messages:
  *Transaction (Process ID %d) was deadlocked on %.*ls resources with another process and has been chosen as the deadlock victim. Rerun the transaction.*
- The *Deadlocks Guide* in Microsoft Learn, lists these resources that can deadlock:
  - Locks. The only one I will talk about.
  - Parallel query execution-related resources.
  - Worker threads.
  - Memory.
  - Multiple Active Result Sets (MARS) resources.

© Erland Sommarskog 2026

Getting Information about Deadlocks

- Easiest is to query the **system_health** session which is always running.

```
SELECT CAST(event_data AS xml), timestamp_utc
FROM sys.fn_xe_file_target_read_file(
     N'system_health*.xel',
     DEFAULT, DEFAULT, DEFAULT)
WHERE object_name = 'xml_deadlock_report'
ORDER BY timestamp_utc DESC
```

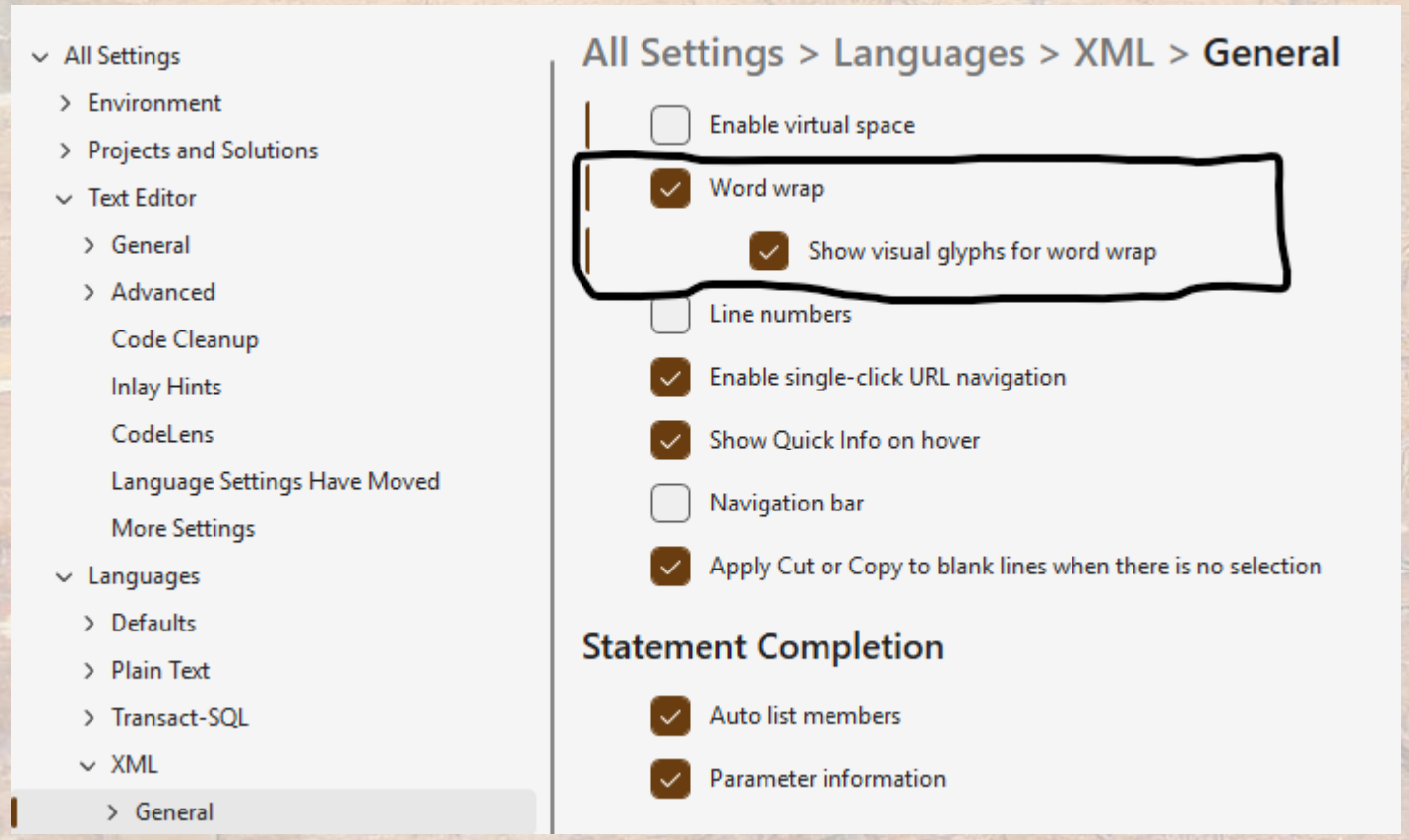
02_query_system_health.sql

- Observe the asterisk – without it you get no output.
- timestamp_utc only available on SQL 2017 and later.

© Erland Sommarskog 2026

Tip for SSMS

- To avoid having to scroll sideways, turn on word wrap for XML:



© Erland Sommarskog 2026

Defining Your Own XEvent Session

- You find that querying system_health session is slow.
- You want low latency when testing.
  - System_health has a latency of two minutes. This is good for production, but maybe not if you produce deadlock at will and want to see the result.
- Use the event **xml_deadlock_report**.

© Erland Sommarskog 2026

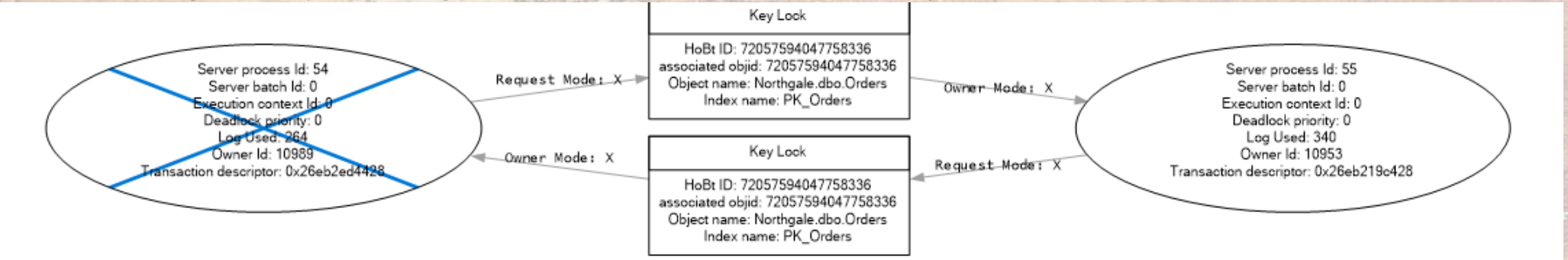
Azure SQL Database

- In Azure SQL Database, system_health is not available.
- There is a built-in dedicated session for deadlocks, but it can cause issues on logical servers with many databases.
- Create XE session writing to BLOB store or ring buffer.
  - Script file has an example query for ring_buffer.
- Use event **database_xml_deadlock_report**.

© Erland Sommarskog 2026

Other Means to Get Information

- Turn on TF 1222 to get XML in the SQL Server errorlog.
  - Simple, but litters the errorlog, and information is difficult to read.
- Trace/Profiler. Mainly an option on SQL 2005/2008.
  - Profiler gives you a graph that looks pretty:



  - But it's confusing and important information is missing. Discard!
  - The TextData column has the actual XML.

© Erland Sommarskog 2026

The Deadlock XML

```
<deadlock>
  <victim-list>
    <victimProcess id="process18519e59088" />
  </victim-list>
  <process-list>...</process-list>
  <resource-list>...</resource-list>
</deadlock>
```

- `victim-list` – The process(es) that were rolled back.
- `process-list` – Information about the processes.
- `resource-list` – The locks involved in the deadlock.

© Erland Sommarskog 2026

The Process List

```
<process-list>
  <process id="process2b79fb44ca8" ... >
      <stackFrames>...</stackFrames>
      <executionStack>...</executionStack>
      <inputbuf>...</inputbuf>
  </process>
  <process id="process2b792576">...</process>
</process-list>
```

- One tag for each process in the deadlock.
- Many attributes about the processes.

© Erland Sommarskog 2026

Nested Elements in the Process Tag

```
<stackFrames>...</stackFrames>
<executionStack>...</executionStack>
<inputbuf>...</inputbuf>
```

- `stackFrames` – SQL Server call stack, only of interest to support engineers. (New in SQL 2022).
- `executionStack` and `inputbuf` – information about the submitted command and current statement.

© Erland Sommarskog 2026

The inputbuf Tag, take one

```
      <inputbuf>
UPDATE dbo.Orders
SET    Freight = 17.23
WHERE  OrderID = 11000
   </inputbuf>
```

- For ad-hoc commands, the `inputbuf` tag often suffices.
- For multi-statement batches, you may have use for the line numbers in the `executionStack` tag.

© Erland Sommarskog 2026

The inputbuf Tag, Take Two

- You may be looking at:

```
      <inputbuf>
Proc [Database Id = 2 Object Id = 757577737]    </inputbuf>
```

- This is a stored procedure called through RPC.
- You need to look at the `executionStack` tag for more information.

© Erland Sommarskog 2026

The executionStack Tag

```
<executionStack>
   <frame procname="tempdb.dbo.inner_sp" line="10" stmtstart="502"
    stmtend="612" sqlhandle="0x03000200ee8...">
INSERT RollYourOwn(id, value) VALUES(@id, @value    </frame>
   <frame procname="tempdb.dbo.outer_sp" line="3" stmtstart="150"
    stmtend="212" sqlhandle="0x0300020027b...">
EXEC inner_sp @value, @id OUTPU    </frame>
</executionStack>
```

- You see the statement of the deadlock, and the call stack.
- In this example: the application called outer_sp.
- Last character in statement is stripped off on SQL 2014+.

© Erland Sommarskog 2026

All the Process Attributes

```
<process id="process26eab72f848" taskpriority="0" logused="244"
waitresource="KEY: 6:72057594047889408 (fadcdcb5e33c)"
waittime="2728" ownerId="4260688" transactionname="UPDATE"
lasttranstarted="2022-01-30T14:57:47.787" XDES="0x270d78b8428"
lockMode="X" schedulerid="1" kpid="9744" status="suspended"
spid="53" sbid="0" ecid="0" priority="0" trancount="2"
lastbatchstarted="2022-01-30T14:57:47.783"
lastbatchcompleted="2022-01-30T14:57:47.783" lastattention="1900-
01-01T00:00:00.783" clientapp="slask.pl" hostname="SOMMERWALD"
hostpid="6632" loginname="PRESENT10\sommar" isolationlevel="read
committed (2)" xactid="4260688" currentdb="6"
currentdbname="Northgale" lockTimeout="4294967295"
clientoption1="671088672" clientoption2="128056">
```

© Erland Sommarskog 2026

The Relevant Ones

```
<process id="process26eab72f848" taskpriority="" logused=""
waitresource="" waittime="" ownerId=""
transactionname="UPDATE"
lasttranstarted="2022-01-30T14:57:47.787" XDES="" lockMode=""
schedulerid="" kpid="" status="" spid="53" sbid="0" ecid="0"
priority="" trancount="2"
lastbatchstarted="2022-01-30T14:57:47.783"
lastbatchcompleted="2022-01-30T14:57:47.783"
lastattention="1900-01-01T00:00:00.783" clientapp="slask.pl"
hostname="SOMMERWALD" hostpid="" loginname="PRESENT10\sommar"
isolationlevel="read committed (2)" xactid="" currentdb=""
currentdbname="Northgale" lockTimeout="" clientoption1=""
clientoption2="">
```

© Erland Sommarskog 2026

Process Identification

```
<process id="process26eab72f848" taskpriority="" logused=""
waitresource="" waittime="" ownerId=""
transactionname="UPDATE"
lasttranstarted="2022-01-30T14:57:47.787" XDES="" lockMode=""
schedulerid="" kpid="" status="" spid="53" sbid="0" ecid="0"
priority="" trancount="2"
lastbatchstarted="2022-01-30T14:57:47.783"
lastbatchcompleted="2022-01-30T14:57:47.783“
lastattention="1900-01-01T00:00:00.783" clientapp="slask.pl"
hostname="SOMMERWALD" hostpid="" loginname="PRESENT10\sommar"
isolationlevel="read committed (2)" xactid="" currentdb=""
currentdbname="Northgale" lockTimeout="" clientoption1=""
clientoption2="">
```

© Erland Sommarskog 2026

Process Identification Details

- `id="process26eab72f848"` Mapping between `process-list`, `resource-list` and `victim-list`.
- `spid="53"` The one we know and love.
- `sbid="0"` Non-zero if MARS is in use.
- `ecid="0"` Non-zero when there is parallelism.
- `currentdbname` and `loginname`. Self-explanatory.
- `clientapp` and `hostname`. Recall that they are set in the connection string.

© Erland Sommarskog 2026

transactionname Attribute

```
<process id="process26eab72f848" taskpriority="" logused=""
waitresource="" waittime="" ownerId=""
transactionname="UPDATE"
lasttranstarted="2022-01-30T14:57:47.787" XDES="" lockMode=""
schedulerid="" kpid="" status="" spid="53" sbid="0" ecid="0"
priority="" trancount="2"
lastbatchstarted="2022-01-30T14:57:47.783"
lastbatchcompleted="2022-01-30T14:57:47.783"
lastattention="1900-01-01T00:00:00.783" clientapp="slask.pl"
hostname="SOMMERWALD" hostpid="" loginname="PRESENT10\sommar"
isolationlevel="read committed (2)" xactid="" currentdb=""
currentdbname="Northgale" lockTimeout="" clientoption1=""
clientoption2="">
```

© Erland Sommarskog 2026

Multi-Statement Transaction?

- In a multi-statement transaction, locks in the deadlock may come from previous statements in the transaction!
- If `transactionname` reads *user_transaction*, you have a multi-statement transaction.
- Same if it is *MyTran* or some other name. (Someone said `BEGIN TRANSACTION MyTran`).
- *implicit_transaction* is also a multi-statement transaction. `SET IMPLICIT_TRANSACTIONS ON` is in effect.

© Erland Sommarskog 2026

Multi-Statement Transaction? cont'd

- If `transactionname` reads SELECT, INSERT, UPDATE, DELETE etc, likely to be an auto-committed statement.
- But check `executionStack` – the deadlock statement could be in a trigger with multiple statements.
  - A trigger always executes in the context of a transaction defined by the statement that fired it.
- `transactionname` can read INSERT EXEC. In this case the procedure runs a multi-statement transaction defined by the INSERT statement.

© Erland Sommarskog 2026

Three Important Timestamps

```
<process id="process26eab72f848" taskpriority="" logused=""
waitresource="" waittime="" ownerId=""
transactionname="UPDATE"
lasttranstarted="2022-01-30T14:57:47.787" XDES="" lockMode=""
schedulerid="" kpid="" status="" spid="53" sbid="0" ecid="0"
priority="" trancount="2"
lastbatchstarted="2022-01-30T14:57:47.783"
lastbatchcompleted="2022-01-30T14:57:47.783"
lastattention="1900-01-01T00:00:00.783" clientapp="slask.pl"
hostname="SOMMERWALD" hostpid="" loginname="PRESENT10\sommar"
isolationlevel="read committed (2)" xactid="" currentdb=""
currentdbname="Northgale" lockTimeout="" clientoption1=""
clientoption2="">
```

© Erland Sommarskog 2026

Transaction Started by Application?

- Is `lasttranstarted` before `lastbatchstarted`? Typically, it's a multi-batch transaction started by the application.
  - But it could be a runaway transaction without a COMMIT.
- Are `lastbatchcompleted` and `lastbatchstarted` far apart in a multi-batch transaction? Investigate!
  - Is it doing work elsewhere which increases the length of the transaction – and thus makes the window for a deadlock wider?

© Erland Sommarskog 2026

The lastattention Attribute

```
<process id="process26eab72f848" taskpriority="" logused=""
waitresource="" waittime="" ownerId=""
transactionname="UPDATE"
lasttranstarted="2022-01-30T14:57:47.787" XDES="" lockMode=""
schedulerid="" kpid="" status="" spid="53" sbid="0" ecid="0"
priority="" trancount="2"
lastbatchstarted="2022-01-30T14:57:47.783"
lastbatchcompleted="2022-01-30T14:57:47.783"
lastattention="1900-01-01T00:00:00.783" clientapp="slask.pl"
hostname="SOMMERWALD" hostpid="" loginname="PRESENT10\sommar"
isolationlevel="read committed (2)" xactid="" currentdb=""
currentdbname="Northgale" lockTimeout="" clientoption1=""
clientoption2="">
```

© Erland Sommarskog 2026

Attention Signals

- The application can send an *attention signal* to SQL Server.
- On this signal, SQL Server will:
  - Roll back the current statement.
  - Leave any user transaction open (unless XACT_ABORT is on).
- Common reason is the error *Timeout Expired*. (Most APIs have a default timeout of 30 s.)
- The red button in SSMS also produces an attention signal.
- (There are a few more less common ways this can happen.)

© Erland Sommarskog 2026

Unhandled Timeout Errors

- `lastattention` reflects the last time an attention signal was received.
- Is `lastattention` later than `lasttranstarted`? This is a red flag for a runaway transaction!
- Most likely the application has experienced a timeout without rolling back.
- Stop analysing the deadlock. Instead, review the error-handling code in the application.

© Erland Sommarskog 2026

Transaction Count

```
<process id="process26eab72f848" taskpriority="" logused=""
waitresource="" waittime="" ownerId=""
transactionname="UPDATE"
lasttranstarted="2022-01-30T14:57:47.787" XDES="" lockMode=""
schedulerid="" kpid="" status="" spid="53" sbid="0" ecid="0"
priority="" trancount="2"
lastbatchstarted="2022-01-30T14:57:47.783"
lastbatchcompleted="2022-01-30T14:57:47.783"
lastattention="1900-01-01T00:00:00.783" clientapp="slask.pl"
hostname="SOMMERWALD" hostpid="" loginname="PRESENT10\sommar"
isolationlevel="read committed (2)" xactid="" currentdb=""
currentdbname="Northgale" lockTimeout="" clientoption1=""
clientoption2="">
```

© Erland Sommarskog 2026

The Deceivable trancount Attribute

- For an INSERT, UPDATE, DELETE or MERGE statement, it will always be at least 2, never one or zero.
- And it's 2, regardless if it is a single-statement transaction or a multi-statement transaction.
- For a nested transaction it is 3 or higher.
- A value that does not match the stack depth could indicate a runaway transaction.
- Transaction name + `trancount` = 0 ➔ System operation.

© Erland Sommarskog 2026

The isolationlevel Attribute

```
<process id="process26eab72f848" taskpriority="" logused=""
waitresource="" waittime="" ownerI
transactionname="UPDATE"
lasttranstarted="2022-01                    787" XDES="" lockMode=""
schedulerid="" kpi                    spid="53" sbid="0" ecid="0"
priority="" tra
lastbatchstarte                    01-30T14:57:47.783"
lastbatchcomplet                    "2022-01-30T14:57:47.783"
lastattention="1900-01-01T00:00:00.783" clientapp="slask.pl"
hostname="SOMMERWALD" hostpid="" loginname="PRESENT10\sommar"
isolationlevel="read committed (2)" xactid="" currentdb=""
currentdbname="Northgale" lockTimeout="" clientoption1=""
clientoption2="">
```

Covered later

© Erland Sommarskog 2026

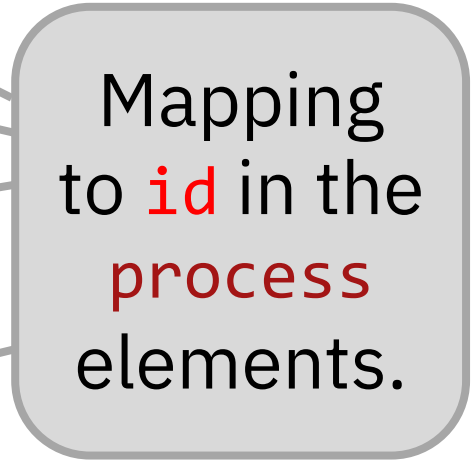
The resource-list Element - Overview

```
<resource-list>
  <keylock attr="...">
    <owner-list>...</owner-list>
    <waiter-list>...</waiter-list>
  </keylock>
  <keylock attr="...">
    <owner-list>...</owner-list>
    <waiter-list>...</waiter-list>
  </keylock>
</resource-list>
```

- One element for each lock in the deadlock.
- Name of element depends on type of lock.
- Nested in that element:
  - `owner-list` element listing the owner(s) of the lock.
  - `waiter-list` element listing processes waiting for the lock.

© Erland Sommarskog 2026

```
<keylock attr="...">
  <owner-list>
    <owner id="process26eabfb68c8" mode="X" />
  </owner-list>
  <waiter-list>
    <waiter id="process26eab72e4e8" mode="S" requestType="wait" />
  </waiter-list>
</keylock>
<keylock attr="...">
  <owner-list>
    <owner id="process26eab72e4e8" mode="S" />
  </owner-list>
  <waiter-list>
    <waiter id="process26eabfb68c8" mode="X" requestType="wait" />
  </waiter-list>
</keylock>
```



© Erland Sommarskog 2026

The xxxlock Element

```
<keylock hobtid="72057594047758336" dbid="6"
objectname="Northgale.dbo.Orders" indexname="PK_Orders"
id="lock2709eb35780" mode="X"
associatedObjectId="72057594047758336">
```

- You need to know table and index name. Ignore the rest.
- `keylock` = Row lock in index, clustered or non-clustered.
- `ridlock` = Row lock in heap when accessing data page.
- `pagelock` = What it says, lock on page level.
- `objectlock` = Lock on table level.

© Erland Sommarskog 2026

Index Name in Page Locks

- In `pagelock` elements, the index name is missing.
- Work from the `associatedObjectId` attribute with this query:

```
SELECT object_name(p.object_id) AS table_name,
       i.name AS index_name
FROM   sys.partitions p
JOIN   sys.indexes i ON p.object_id = i.object_id
                    AND p.index_id = i.index_id
WHERE  p.hobt_id = @associatedObjectId
```

© Erland Sommarskog 2026

Advanced: Which Row is Locked?

- Look in the `process` element for the waiter:

```
waitresource="KEY: 6:72057594047758336
(3b86dcc184bf)"
```

- Copy the parenthetical part and run:

```
SELECT * FROM tbl WITH (INDEX = index_name)
WHERE %%LOCKRES%% = '(3b86dcc184bf)'
```

- Must force the index in the `keylock` element!

© Erland Sommarskog 2026

Deadlocks with Parallelism

- If a deadlock occurs with parallel execution, there is one `process` per thread with same `spid` and different `ecid`.
- The `resource-list` is likely to have `exchangeEvent` elements, because threads are waiting for each other.
- Try to ignore these, and focus on the locks.
- If all processes in the `process-list` have the same `spid`, you have an intra-query deadlock.
  - They are bugs in SQL Server.

© Erland Sommarskog 2026

Quick Recap on Locking

- Granularity.
- The most important lock types.
- Intent locks.
- (Optimized locking. We come to this later.)

© Erland Sommarskog 2026

Recap on Locking - Granularity

- For DML operations, locks can be taken on Row, Page or Table level.
  - (And partition, if enabled for the table.)
- The higher the level, the bigger the risk for deadlocks.
- Normally, SQL Server takes row locks, but if it thinks that a query can touch many rows, it may start with page or table locks to preserve memory.

© Erland Sommarskog 2026

Lock Escalation

- If a *query* takes more than ~5000 locks on a table, SQL Server attempts to escalate to table level.
- If escalation is blocked by other locks, SQL Server jogs along with the row locks and tries again later.
- Lock escalation is never to page level.
- Thus, if you see page locks in a deadlock, they were there when the query started.

© Erland Sommarskog 2026

Controlling Lock Escalation

- You can use ALTER TABLE SET LOCK_ESCALATION to control lock escalation per table.
  - TABLE – the default, table level.
  - AUTO – escalation to partition, if table is partitioned.
  - DISABLE – What it says.
- Trace Flags
  - 1224 – Disable lock escalation, unless there is memory pressure.
  - 1211 – Disable lock escalation, period.

© Erland Sommarskog 2026

The Basic Lock Modes

From strongest to weakest:

- Sch-M – Schema-modification locks.
- X – Exclusive locks.
- U – Update locks.
- S – Shared locks.
- Sch-S – Schema-stability locks.

© Erland Sommarskog 2026

Schema-Modification Locks (Sch-M)

- Always on table level.
- Taken by DDL statements (ALTER TABLE, CREATE INDEX etc.)
- Incompatible with all other locks.
- Held to the end of transaction.

© Erland Sommarskog 2026

Exclusive Locks (X)

- Can be on any level.
- Taken by INSERT, UPDATE, DELETE and MERGE.
- Incompatible with all other locks but Sch-S.
- Held to the end of transaction.

© Erland Sommarskog 2026

Schema-Stability Locks (Sch-S)

- Always on table level.
- Taken by SELECT and other read operations in:
  - Snapshot isolation.
  - Read-committed snapshot.
  - Read uncommitted, a.k.a. NOLOCK.
- Compatible with all locks, but Schema-modification.
  - Prevents schema from changing while query is running.
- Held to the end of the query.
- Also taken during compilation of a query.

© Erland Sommarskog 2026

Shared Locks (S)

- Can be on any level.
- Taken by SELECT and other read operations with:
  - Read Committed without snapshot.
  - Repeatable Read.
  - Serializable.
- Incompatible with Sch-M and Exclusive locks.
- Compatible with Sch-S, Update locks and other Shared locks.
- Held as long as needed by the isolation level.

© Erland Sommarskog 2026

Update Locks (U)

- Can be taken on any level.
- Taken by UPDATE, DELETE and MERGE.
- Compatible with Sch-S and Shared locks.
- Incompatible with Sch-M, Exclusive *and other* Update locks.
- Taken when checking if a row matches the conditions.
- Converted to Exclusive or released when not matching.

© Erland Sommarskog 2026

Why Update Locks?

```
UPDATE tbl SET ... WHERE nonindexedcol = @value
```

Read a row to see if it qualifies, how to do that?

- Take an exclusive lock in case we need to update?
  - Blocks readers!
- Take a shared lock to prevent other updates?
  - If two processes read in parallel, they will block each other from converting to Exclusive. (I.e. a deadlock!)
- Update locks solve this problem.

© Erland Sommarskog 2026

Intent Locks

- A.k.a “I am here” locks.
- A process wants an exclusive lock on table level. This should be blocked if there is a shared lock on any row.
- Check all rows to see if any is locked?
- That would be highly inefficient!
- Intent locks to the rescue!

© Erland Sommarskog 2026

Intent Locks at Work

- Consider: `SELECT Col1, Col2 FROM tbl WHERE keycol = 987.`
- First take an Intent Shared (IS) lock on table level.
- This will be blocked if another process holds a schema-modification (Sch-M) or exclusive (X) lock on table level.
- Once the IS lock is taken, any attempt to take an exclusive table lock will be blocked.
- Repeat on page level.
- Finally, take a Shared (S) lock the row with keycol = 987.
- Only need to check for locks on the resource itself.

© Erland Sommarskog 2026

Intent Locks and Compatibility

- There are three types of Intent Locks:
  - Intent Shared (IS).
  - Intent Update (IU).
  - Intent Exclusive (IX).
- All intent locks are compatible with each other.
- IS is incompatible with X and Sch-M.
- IU is incompatible with U, X and Sch-M.
- IX is incompatible with S, U, X and Sch-M.

© Erland Sommarskog 2026

Isolation Levels

- Read Uncommitted.
- Read Committed (the default, has two implementations).
- Snapshot.
- Repeatable Read.
- Serializable.

© Erland Sommarskog 2026

Controlling Isolation Levels

- There are two ways.
- By SET command, for instance:

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
```

  - This level applies to all queries until there is a new SET TRANSACTION ISOLATION LEVEL command or module exits.
- By hint for a specific table in a query, for instance:

```
SELECT * FROM tbl WITH (SERIALIZABLE)
```

© Erland Sommarskog 2026

Read Uncommitted

- Semantics: you accept to read data that is uncommitted and which could be inconsistent.
- Also known as “dirty reads”.
- In query hints often specified as NOLOCK.

© Erland Sommarskog 2026

Read Uncommitted, Semantics

- Consider this data and query:

```
SELECT SUM(Amount)
FROM   tbl
WHERE  Customer = 'Jane'
```

Customer	Account	Amount
Jane	123	100
Jane	234	200
Jane	345	300
Jane	456	400

- As query runs, a transaction books a transfer of 50€ from account 234 to 345.
- Our query may return 950, 1050 or something else depending on how the transfer is implemented.

© Erland Sommarskog 2026

Read Uncommitted, implementation

- SELECT statements do not take shared locks on row, page or table level.
- They do take Sch-S locks for all tables in the query.
- Thus, you can still be blocked by DDL, for instance index rebuild.
- INSERT, UPDATE, DELETE and MERGE will take Update/Exclusive locks for their target tables.

© Erland Sommarskog 2026

NOLOCK and Deadlocks

- Sure, NOLOCK can remove plenty of deadlocks.
- ...and your queries will start to produce spurious incorrect results that are difficult to reproduce.
- With NOLOCK you may:
  - Read uncommitted data.
  - Fail to read committed data that was moved due to page splits.
  - Read the same data twice.
  - Read incomplete LOB values.
- NOLOCK is not a solution to deadlocks, period!
- We will look at a better alternative later.

© Erland Sommarskog 2026

Read Committed

- Semantics: Guaranteed to read only committed data.
- Thus the query

```
SELECT SUM(Amount) FROM tbl WHERE Customer = 'Jane'
```

  can only return 1000, even if a transfer is in progress.
- Say that while query is running, Jane withdraws 150€ from account 456 the ATM.
- Our query may return 850 or 1000. Both are correct under Read Committed.
- If we run the query twice in a transaction, we could get 1000 the first time and 850 the second time.

© Erland Sommarskog 2026

Read Committed in SQL Server

- This is the default isolation level in SQL Server.
- There are two implementations: Locks and Snapshot.
- Locks is the default in the box product and Azure SQL Managed Instance.
- Snapshot is the default in Azure SQL Database.
- Foreign-key validations always use locks.

© Erland Sommarskog 2026

Configuring Read Committed

- To enable/disable Read Committed Snapshot:

```
ALTER DATABASE db SET READ_COMMITTED_SNAPSHOT ON / OFF
```

  - Requires exclusive access to the database.
- When disabled, Read Committed is always by Locks.
- When enabled, Read Committed defaults to Snapshot.
- Can be overridden by READCOMMITTEDLOCK hint.
  - There is no SET TRANSACTION ISOLATION LEVEL command.

© Erland Sommarskog 2026

Read Committed with Locks

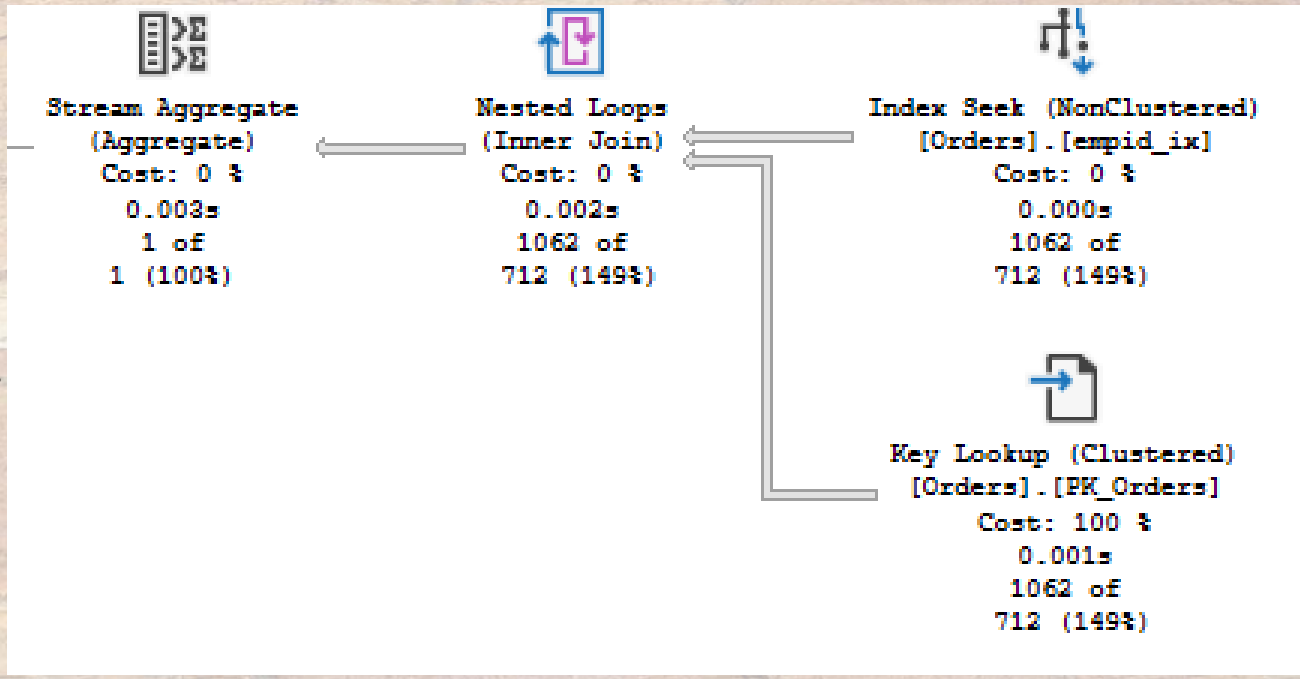
- You read directly from the data rows, taking shared locks. (On row, page or table.)
- Thus, you are blocked by uncommitted updates.
- Locks are released when not needed anymore, often before the end of the query.
- Hm, what does that mean exactly?

© Erland Sommarskog 2026

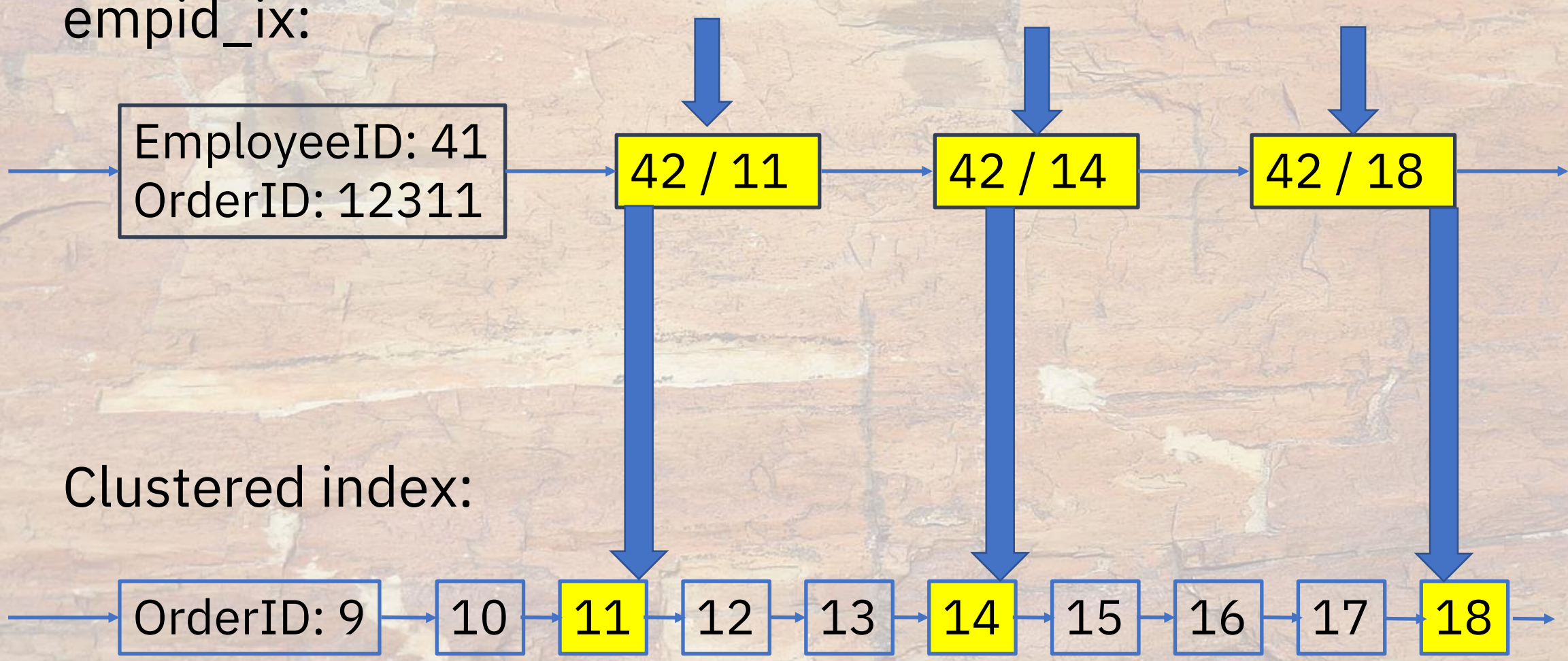
Example: When Not Needed Anymore?

```
SELECT SUM(Freight) FROM Orders WHERE EmployeeID = 42
```

Execution plan:
Index Seek + Key Lookup.



© Erland Sommarskog 2026



© Erland Sommarskog 2024

Read Committed Snapshot (RCSI)

- SQL Server reads data from a version store, not from the data rows.
  - Except for target tables for UPDATE, DELETE and MERGE.
- A statement sees a snapshot of the database as it was when the statement started. (...ish)
  - User-defined functions have their own starting points.
- If a row has an uncommitted change, you get the old data.
  - `SELECT SUM(Amount) FROM tbl WHERE Customer = 'Jane'` runs while Jane withdraws 150€. The result will be 1000, the sum before the withdrawal.

© Erland Sommarskog 2026

RCSI, the Deadlock Killer

- With RCSI no shared locks are taken on rows, only Sch-S locks on table level.
- Thus, with RCSI readers cannot block writers and writers cannot block readers.
- Enabling Read Committed Snapshot is great for removing deadlocks between readers and writers with no code changes!

© Erland Sommarskog 2026

RCSI, Technical Caveats

- There is a price: Incurs an overhead to update and delete operations to maintain the version store.
- Version store is in tempdb (unless Accelerated Database Recovery is turned on).
  - Thus, you may have increase size of tempdb.
- Adds a 14-byte pointer to updated rows, leading to page splits.
  - These pointers are removed on index rebuild!
  - Rebuild indexes with 90% fill factor?

© Erland Sommarskog 2026

RCSI, Application Caveats

- You are reading stale data. Does that matter?
  - Example: Cannot inactivate a customer with an active order. An inactivated customer cannot add a new order. This is checked by triggers.
  - Transaction A: `INSERT Orders(CustomerID, ...) VALUES(77, ..)`
  - Transaction B: `UPDATE Customers SET Active = 0 WHERE ID = 77`
  - Trigger on Orders reads from snapshot and sees that customer is active.
  - Trigger on Customers reads from snapshot and sees no active order.
  - Business-rule violation!
- Generally, post/pre-update validations do best in using READCOMMITEDLOCK hint.
  - Foreign-key validations is always by locks.

© Erland Sommarskog 2026

Snapshot Isolation

- What you can use if RCSI makes you nervous.
- In true snapshot isolation, you see the database as it was when then transaction started.
  - No …ish this time!
- Thus, our sum query *must* return 1000 when Jane is at the ATM.
- All reads are from the version store.
  - Including target tables for UPDATE, DELETE and MERGE.
- No shared locks taken, only Sch-S locks on tables.

© Erland Sommarskog 2026

Snapshot Isolation, cont'd

- By default, snapshot isolation is not available.
- You need to enable it with

```
ALTER DATABASE db SET ALLOW_SNAPSHOT_ISOLATION ON
```

  - Does not require exclusive database access.
- To run a snapshot transaction, you need to say

```
SET TRANSACTION ISOLATION LEVEL SNAPSHOT
```

- Because you need to buy in on snapshot isolation, you can control where it is used.

© Erland Sommarskog 2026

Repeatable Read

- Semantics: if you read the same row twice in a transaction, you will get back the same result.
- Use case: Say that we are running
  ```
  SELECT SUM(Amount) FROM tbl WHERE Customer = 'Jane'
  ```
  We intend to use those 1000€ later in the transaction, so we want to make sure that result does not change.
- Shared locks are held to the end of the transaction.
  - Locks on rows filtered out are released directly.

© Erland Sommarskog 2026

Serializable

- With Repeatable Read, deposits or withdrawals to the existing accounts will be blocked.
- What if someone adds a new account, for instance a loan with a negative balance?
- With Serializable we are guaranteed that rows can not be added to a range we have read.
- Rather than plain row locks, SQL Server uses key-range locks, for instance RangeS-S, to implement Serializable.

© Erland Sommarskog 2026

Reading a Value to Update it Later, I

```
SELECT @val = col FROM tbl WHERE id = @id
SELECT @val += @incr
UPDATE tbl SET col = @val WHERE id = @id
```

Transaction A, @id=56, @incr=5.

- Reads current value of col, 10.
- Releases shared lo
- Computes @val = 1
- Updates row to 15 and commits.

Transaction B, @id=56, @incr=7.

- R nt value of col, 10.
- hared lock.
- Computes @val = 17.
- Updates row to 17 and commits.

WRONG RESULT!

© Erland Sommarskog 2026

Reading a Value to Update it Later, II

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
SELECT @val = col FROM tbl WHERE id = @id
SELECT @val += @incr
UPDATE tbl SET col = @val WHERE id = @id
```

Transaction A, @id=56, @incr=5.

- Reads current value
  and retains lock.
- Computes @val = 15.
- Tries to update row, but is blocked by B.

DEADLOCK!

Transaction B, @id=56, @incr=7.

- Reads current value of col, 10
  lock.
- val = 17.
- Tries to update row, but is blocked by A.

© Erland Sommarskog 2026

Reading a Value to Update Later, III

```
SELECT @val = col FROM tbl WITH (UPDLOCK) WHERE id = @id
SELECT @val += @incr
UPDATE tbl SET col = @val WHERE id = @id
```

Transaction A

- Reads current value of col, 10, retains Update lock
- Computes @val = 15.
- Updates row to 15.
- Commits and releases lock.

CORRECT!

Transaction B

- Tries to read row, but is
- Reads 15.
- Computes @val = 22.
- Updates row to 22, commits.

© Erland Sommarskog 2026

The Power of UPDLOCK

- When reading a value to update it later, use UPDLOCK to

1. Prevent that another process changes the row.
2. Prevent deadlocks.

- UPDLOCK implies Repeatable Read.
- When you read a range that you will update, you need to say WITH (UPDLOCK, SERIALIZABLE).

© Erland Sommarskog 2026

The isolationlevel Attribute

```
<process id="process26eab72f848" taskpriority="" logused=""
waitresource="" waittime="" ownerId=""
transactionname="UPDATE"
lasttranstarted="2022-01-30T14:57:47.787" XDES="" lockMode=""
schedulerid="" kpid="" status="" spid="53" sbid="0" ecid="0"
priority="" trancount="2"
lastbatchstarted="2022-01-30T14:57:47.783"
lastbatchcompleted="2022-01-30T14:57:47.783"
lastattention="1900-01-01T00:00:00.783" clientapp="slask.pl"
hostname="SOMMERWALD" hostpid="" loginname="PRESENT10\sommar"
isolationlevel="read committed (2)" xactid="" currentdb=""
currentdbname="Northgale" lockTimeout="" clientoption1=""
clientoption2="">
```

© Erland Sommarskog 2026

The isolationlevel Attribute

- Should always say Read Committed (or Snapshot).
- That is, don't use SET TRANSACTION ISOLATION LEVEL for Repeatable Read and Serializable.
- Force higher isolation levels with hints for the tables where transaction semantics calls for it.
- For other tables, permits locks to be released earlier – or use the snapshot.
- Beware: some transaction APIs turn on Serializable by default, so it may be on "by mistake".

© Erland Sommarskog 2026

Preventing Deadlocks

- READ_COMMITTED_SNAPSHOT. (Already covered)
- The UPDLOCK hint. (Already covered)
- Always access resources in the same order.
- Query tuning and indexing.
- Turn on Optimized Locking.
- Review application behaviour.
- Serialise with application locks.
- Deadlock priorities and lock timeouts.

© Erland Sommarskog 2026

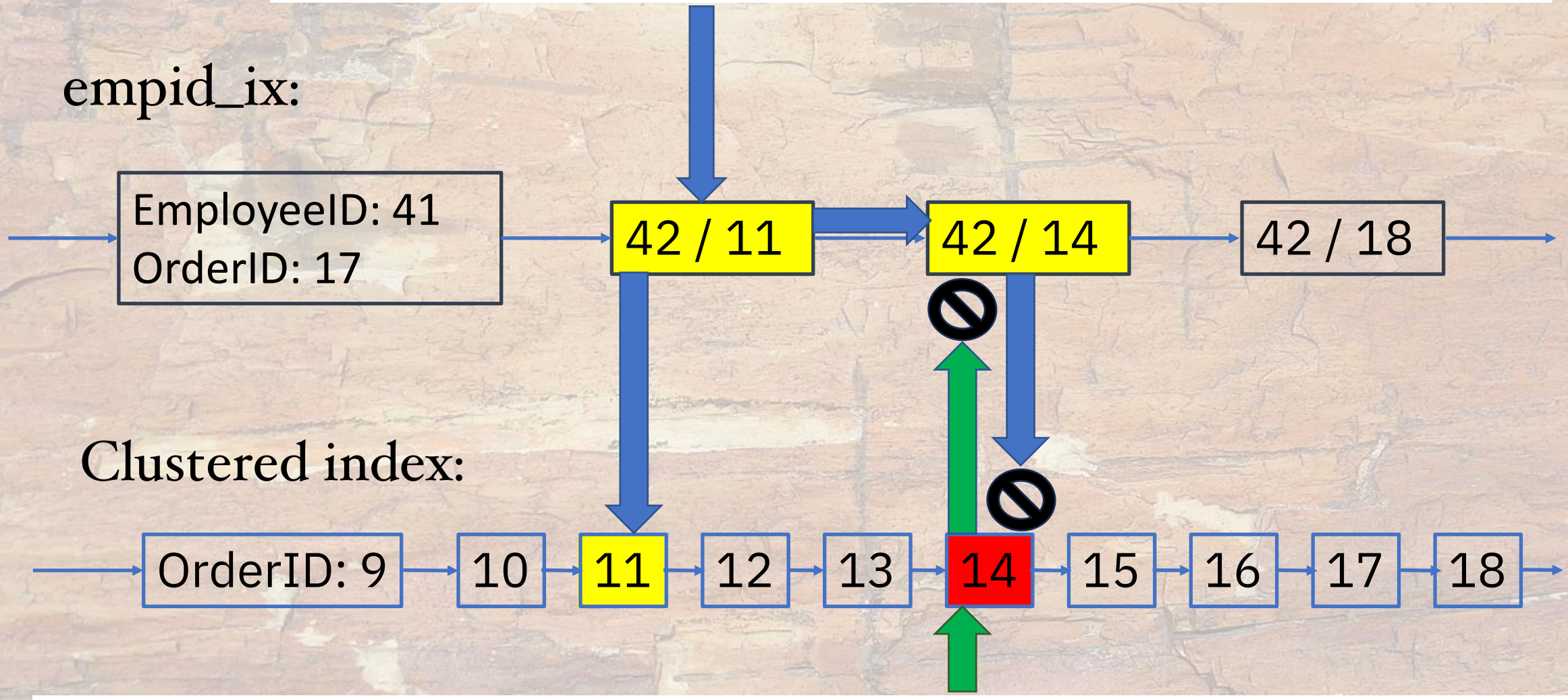
Access Resources in the Same Order

- Standard recommendation. Sounds simple and obvious.
- Can be very difficult to implement in practice.
- Business rules may mandate different access order.
- SQL Server itself does not obey to this rule when building query plans.

Second_window.sql
differentaccessorder.xml

© Erland Sommarskog 2026

```
SELECT SUM(Freight) FROM Orders WHERE EmployeeID = @empid
```




```
UPDATE dbo.Orders SET EmployeeID = @newempid WHERE OrderID = @orderid
```

© Erland Sommarskog 2023

Preventing Deadlocks

- READ_COMMITTED_SNAPSHOT.
- The UPDLOCK hint.
- Always access resources in the same order.
- Query tuning and indexing.
- Turn on Optimized Locking.
- Review application behaviour.
- Serialise with application locks.
- Deadlock priorities and lock timeouts.

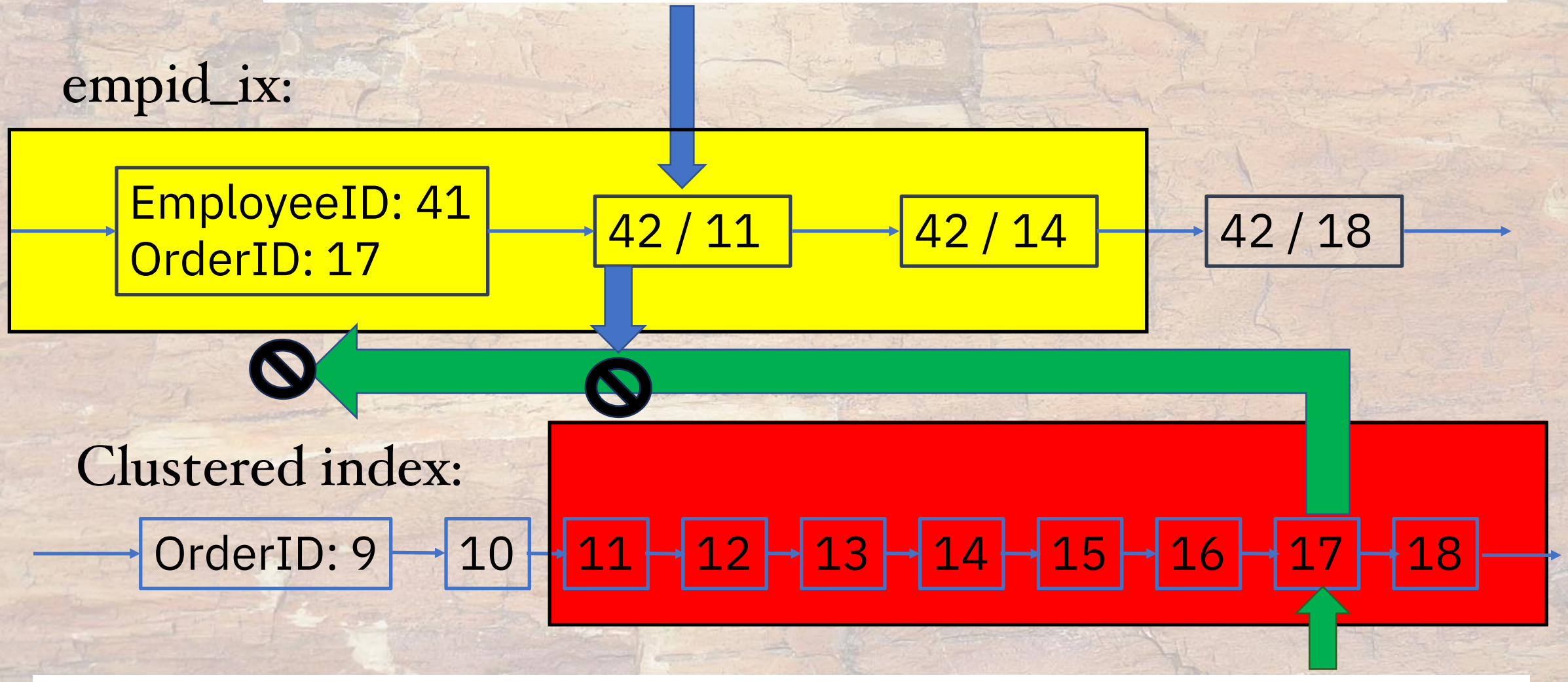
© Erland Sommarskog 2026

The Impact of Page and Table Locks

- In the deadlock we just looked at, the processes were fighting about the very same row.
- But imagine a world with only page locks...

© Erland Sommarskog 2026

```
SELECT SUM(Freight) FROM Orders WHERE EmployeeID = @empid
```




```
UPDATE dbo.Orders SET EmployeeID = @newempid WHERE OrderID = @orderid
```

© Erland Sommarskog 2023

The Impact of Page and Table Locks

- Lower lock granularity increases the risk for deadlocks, as you now can get conflicts on *unrelated* rows.
- Thus, removing such locks helps to prevent deadlocks.
- Recall that intent locks are always on page/table. They are not a problem.
- It's shared/update/exclusive locks on higher levels you should go after.
- They are often a token of a query that needs tuning.

© Erland Sommarskog 2026

Removing Page and Table Locks

- Say that you see this query in a deadlock:

```
SELECT * FROM Orders
WHERE dateadd(MONTH, 1, OrderDate) > getdate()
```

- Entangling column in an expression, prevents Index Seek.
- This leads to Index Scan which often results in page locks.
- No need to analyse the rest of the deadlock, but rewrite:

```
SELECT * FROM Orders
WHERE OrderDate > dateadd(MONTH, -1, getdate())
```

© Erland Sommarskog 2026

Deadlock with Page Locks

Second_window.sql
pagelock-deadlock.xml

```
-- Reader.
SELECT SUM(Freight) FROM TallOrders WHERE OrderDate = @date
-- Writer.
UPDATE TallOrders SET ShipVia = @newshipid WHERE OrderID = @orderid
```

- The `associatedObjectId` in theXML is for an index on (ShipVia, OrderDate).
- Reader scans this index, taking page locks.
- Parallel plan, thus multiple pages locked simultaneously.

© Erland Sommarskog 2026

Deadlock with Page Locks, cont'd

- Writer needs to modify *two* pages in index on ShipVia, one for the old value, one for the new value.
- Thus, writer needs intent locks (IX) on both pages.
- Locks the first page, tries to lock the next but is blocked by a reader thread.
- Another reader thread wants to read rows from first page, but is blocked by writer.
- Possible resolution: add an index on OrderDate.

© Erland Sommarskog 2026

What about the ROWLOCK hint?

- Can be a temporary measure until you get a better solution in place.
- SQL Server chooses page or table locks to preserve memory. Forcing row locks may threaten server stability.
- Therefore, better to review indexes and/or tune queries.
- The ROWLOCK hint does not prevent lock escalation.

© Erland Sommarskog 2026

Are Row Locks Permitted?

- If you have table or page locks you cannot understand, run this query:

```
SELECT * FROM sys.indexes
WHERE  object_name(object_id) = 'YourTable'
  AND (allow_row_locks = 0 OR allow_page_locks = 0)
```

- If it returns any rows, enable page and row locks with:

```
CREATE INDEX ix ON tbl(col) WITH
  (ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
   DROP_EXISTING = ON)
```

© Erland Sommarskog 2026

Only Row Locks, All Good?

- Even if there are only row locks, query tuning can still be the solution.
- Recall this deadlock:

```
-- Reader
SELECT SUM(Freight) FROM Orders WHERE EmployeeID = @empid
-- Writer
UPDATE Orders SET EmployeeID=@newempid WHERE OrderID=@orderid
```

- Possible resolution: Add Freight as an included column in index on EmployeeID.

© Erland Sommarskog 2026

One More Deadlock Case

Consider this:

1. Process P starts a transaction.
2. P updates row(s) in table T.
3. P runs query to populate a temp table for two minutes.
4. Process Q starts SELECT joining S and T, starting from S.
5. The rows in T are updated by P, so Q is blocked.
6. P attempts to update row in S that Q has locked.
7. DEADLOCK!

© Erland Sommarskog 2026

The Window for a Deadlock

- What if we can tune P's INSERT query so that in runs 200 ms?
- That does not entirely *prevent* the deadlock.
- But the *window* where the deadlock can occur is drastically reduced.
- Shorter transactions → Less risk for deadlocks.
- ...but that does not mean you should split up a transaction that needs to be atomic in multiple.

© Erland Sommarskog 2026

Too Many Indexes?

- The more indexes there are on a table, INSERT/ DELETE/ UPDATE statements takes longer. → The deadlock window gets wider.
- Two DML statements that update indexes in different order may clash.
- You may have unused or redundant indexes.
- Use Brent Ozar's sp_BlitzIndex to identify them.

© Erland Sommarskog 2026

Preventing Deadlocks

- READ_COMMITTED_SNAPSHOT.
- The UPDLOCK hint.
- Always access resources in the same order.
- Query tuning and indexing.
- Turn on Optimized Locking.
- Review application behaviour.
- Serialise with application locks.
- Deadlock priorities and lock timeouts.

© Erland Sommarskog 2026

Optimized Locking

- First introduced in Azure SQL Database - 2023.
- Available in SQL Server from SQL 2025.
- Two parts: TID locking and Lock After Qualification.
- Requires Accelerated Database Recovery (ADR) to be on.
- ADR and Optimized Locking are always enabled in Azure SQL Database and Azure SQL Managed Instance.
- Opt-in features in SQL Server on-prem.
  - Need exclusive database access to change the settings.

© Erland Sommarskog 2026

Accelerated Database Recovery

- Introduced in SQL 2019.
- Aims at reducing the recovery time for a database at startup.
- Helps to keep down the transaction log in size.
- Maintains PVS, Persistent Version Store, inside the database.
- PVS is also used for RCSI and snapshot isolation.
- Adds overhead to write operations, so it is a trade-off whether to turn it on.

© Erland Sommarskog 2026

TID Locking

- With ADR, a transaction ID (TID) stamped on each row.
- TID Locking: locks on row/page are taken but released immediately and replaced by a lock on the TID, held to the end of transaction.
  - Intent locks on page/table are held to end of transaction.
- Reduces the risk for lock escalation drastically.
- Thus, TID locking can help to avoid deadlocks due to lock escalation.
- Applies mainly to Read Committed isolation level.

© Erland Sommarskog 2026

Lock After Qualification, LAQ

- Only in force when READ_COMMITTED_SNAPSHOT is enabled.
- When locating rows to update, SQL Server uses the version store, no Update locks are taken.
  - That is, the same as in snapshot isolation.
  - If row qualifies, X lock on row (replaced by TID lock).
- Reduces blocking when you have non-indexed filter conditions.
  - And, thus, can prevent deadlocks.

03_optlocking_main.sql
03_optlocking_demo1.sql
03_optlocking_demo2.sql
04_LAQ-Demo1.sql
04_LAQ-Demo2.sql

© Erland Sommarskog 2026

Not a Panacea

- Optimized Locking does not solve all deadlocks.
- None of the deadlocks we looked at earlier are helped by Optimized Locking.
- Main benefit of TID locking is reducing locking memory and avoiding lock escalation.
- LAQ reduces blocking with non-indexed conditions and increases concurrency.
- Reducing deadlocks is mainly a by-effect.

© Erland Sommarskog 2026

Preventing Deadlocks

- READ_COMMITTED_SNAPSHOT.
- The UPDLOCK hint.
- Always access resources in the same order.
- Query tuning and indexing.
- Turn on Optimized Locking.
- Review application behaviour.
- Serialise with application locks.
- Deadlock priorities and lock timeouts.

© Erland Sommarskog 2026

Think of What You Can Do Differently

- Is application running a multi-batch transaction with a lot work between the batches?
- Rewrite one-by-one processing into set-based.
- If two operations clash, can one be rescheduled to a different time?
- Disable user updates when you need to run a big import during business hours.
- Split up the transaction into several in a *controlled* way. (Not a step to take lightly!)

© Erland Sommarskog 2026

Preventing Deadlocks

- READ_COMMITTED_SNAPSHOT.
- The UPDLOCK hint.
- Always access resources in the same order.
- Query tuning and indexing.
- Turn on Optimized Locking.
- Review application behaviour.
- Serialise with application locks.
- Deadlock priorities and lock timeouts.

© Erland Sommarskog 2026

Application Locks

- When you have an operation that cannot run in parallel without clashes, you can serialise with application locks.

```
EXEC sp_getapplock @Resource = 'MyLock',
                   @LockMode = 'Exclusive'
```

  - *MyLock* = name of your lock. Up to 32 characters.
  - Name is local to the database.
  - Blocks until lock is available.
  - Lock is held until end of the transaction.

© Erland Sommarskog 2026

Application Locks, cont'd

- Use application locks when parallel processing should not occur.
- But don't use application locks to prevent occasional deadlocks – can hamper concurrency in the system.
- Application locks can also be taken on session level. Great if you want to prevent parallel execution of a stored procedure entirely.
- See further Microsoft Learn.

© Erland Sommarskog 2026

Preventing Deadlocks

- READ_COMMITTED_SNAPSHOT.
- The UPDLOCK hint.
- Always access resources in the same order.
- Query tuning and indexing.
- Turn on Optimized Locking.
- Review application behaviour.
- Serialise with application locks.
- Deadlock priorities and lock timeouts.

© Erland Sommarskog 2026

Deadlock Priority

- If a background process clashes with end users, you may prefer the background process to be the deadlock victim.
- Have the background process submit this command:

```
SET DEADLOCK_PRIORITY LOW
```

- Rather than LOW, you can give HIGH, NORMAL or a number from -10 to 10. LOW is the same as -5.
- I've never had use for anything but LOW.

© Erland Sommarskog 2026

SET LOCK_TIMEOUT

```
SET LOCK_TIMEOUT 100
```

- If blocked by a lock, wait 100 ms to get the lock.
  - 0 = don't wait. -1 = wait forever.
- When timeout expires, error 1222 is raised.

```
Msg 1222, Level 16, State 51, Line 3
Lock request time out period exceeded.
```

- On this error, rollback and retry (after a short wait).
- Can this be used to prevent deadlocks?

© Erland Sommarskog 2026

Lock Timeouts to Prevent Deadlocks?

- You may have identified that when a process is blocked, that often leads to deadlock.
- One example: Reading from a queue table.
- Could be used by background processes to not hold up more important processes.
  - But not all blocking leads to deadlocks, used too frivolously, so the background process may never get work done.
- Very advanced: There is a NOWAIT table hint, so you can have a lock timeout of 0 on a single table.

© Erland Sommarskog 2026

Priority and timeout in the XML

```
<process id="process26eab72f848" taskpriority="5" logused="244"
waitresource="KEY: 6:72057594047889408 (fadcdcb5e33c)"
waittime="2728" ownerId="4260688" transactionname="UPDATE"
lasttranstarted="2022-01-30T14:57:47.787" XDES="0x270d78b8428"
lockMode="X" schedulerid="1" kpid="9744" status="suspended"
spid="53" sbid="0" ecid="0" priority="-5" trancount="2"
lastbatchstarted="2022-01-30T14:57:47.783"
lastbatchcompleted="2022-01-30T14:57:47.783" lastattention="1900-
01-01T00:00:00.783" clientapp="slask.pl" hostname="SOMMERWALD"
hostpid="6632" loginname="PRESENT10\sommar" isolationlevel="read
committed (2)" xactid="4260688" currentdb="6"
currentdbname="Northgale" lockTimeout="4294967295"
clientoption1="671088672" clientoption2="128056">
```

© Erland Sommarskog 2026

Retry on Deadlock

- Rather than just displaying an error and quit, we can re-attempt the operation that caused the deadlock.
- Some people say: Always retry on deadlock.
- I say: *Only* implement retry in situations where deadlocks are a real problem.
- Incorrectly implemented deadlock retry can lead to data errors that are difficult to explain.
- Testing is difficult – you need a deadlock to test.

© Erland Sommarskog 2026

Two Key Rules for Deadlock Retries

- Never redo only part of a transaction.
  - Never do deadlock retry when called inside a transaction.
    - On deadlock, the *entire* transaction must be rolled back – you cannot only roll back your own part.
- Never redo a committed transaction. That could lead to data being doubled.
  - E.g.: after the transaction there is a SELECT that deadlocks.
- These sort of errors occur only very occasionally and can be very difficult to troubleshoot.

05_deadlockdemo_main.sql
05_deadlockdemo2.sql
05_deadlockdemo3.sql

© Erland Sommarskog 2026

One More Key Rule

- A deadlock retry requires a loop.
- Local variables must be reset at the top of the loop to the value they had before the loop.
- This includes table variables, they are not affected by the rollback!
- Failure to observe this rule can lead to interesting effects, when a variable retains the value from a previous attempt.

© Erland Sommarskog 2026

Code Clutter

- Notice how the deadlock retry clutters the code.
- If you don't want the SELECT inside the transaction, you need a new retry loop for this statement. → Even more clutter.
- Difficult to see the business logic forest behind all those deadlock-retry trees.
- Thus, code is more difficult to maintain and more prone to bugs.

© Erland Sommarskog 2026

Generic Retry in Data Layer?

- Avoids the code clutter – but opens trapdoors.
- You must still check for open transactions.
- How do you know that all stored procedures run at most one transaction? What if there is code after COMMIT that can deadlock?
- Strict coding guidelines?
- Retry of read-only operations is OK, if you can discern them.

© Erland Sommarskog 2026

Retry in the Business Layer?

- If this is where transaction starts, you have no choice.
- You may be back to the code clutter.
- Then again, if you have a background task that runs once a minute – there is an automatic retry.
- And for that matter, telling the user to try again is also a retry…

© Erland Sommarskog 2026

Summary: Key Points of this Session

- An occasional deadlock is no cause for alarm.
- Get the deadlock XML from the system_health session.
- Is any process in a multi-statement transaction? The transaction name will tell you.
- Is any process running a multi-batch transaction? Is `lasttranstarted` before `lastbatchstarted`?
- Remember: The faster your operations are, the smaller the window where a deadlock can occur.
- Only implement deadlock retry if there is no other way out.

© Erland Sommarskog 2026

Summary: Don't Shy the Shortcuts!

- Understanding a deadlock in full detail can be hard.
- The good news: you only need to understand as much to make a decision on how to resolve it.
  - Is `lastattention` after `lasttranstarted`? Troubleshoot the unhandled timeout.
  - When you see page and table locks, apply query and index tuning before anything else.
  - Use RCSI / snapshot isolation to isolate readers and writers.
  - Use deadlock priority + deadlock retry for background processes.
  - Use UPDLOCK to avoid conversion deadlocks.

© Erland Sommarskog 2026

Concurrency Conflicts in Databases

- Deadlocks is just one way they can manifest.
  - Update errors with snapshot isolation or In-Memory OLTP.
  - Or the PK error in the demo.
- Bad throughput due to too strict isolation level or serialisation in the wrong place.
- Worst of them all: Spurious data errors.
  - Too liberal isolation level (which could be RCSI).
  - Incorrect transaction scope.
  - Incorrect deadlock retry.

© Erland Sommarskog 2026

Summary: Choose Wisely

- There is often more than one way to prevent or mitigate a deadlock.
- You need to choose wisely.
- The cure can be worse than the disease.
  - Spurious data errors.
  - Serialisation reducing throughput of the system.
  - Etc.
- Take a holistic view of the situation.

© Erland Sommarskog 2026

The Last Slide

Erland Sommarskog,
esquel@sommarskog.se

Slides and scripts on
http://www.sommarskog.se/present.

To build Northgale:
Run Northwind.sql and then Northgale.sql.

© Erland Sommarskog 2026