cegeka

Read Isolation & Locking

Raimond

NOV 2023

Isolation Levels

Tri State Locking

Optimized Locking

cegeka

Isolation Levels

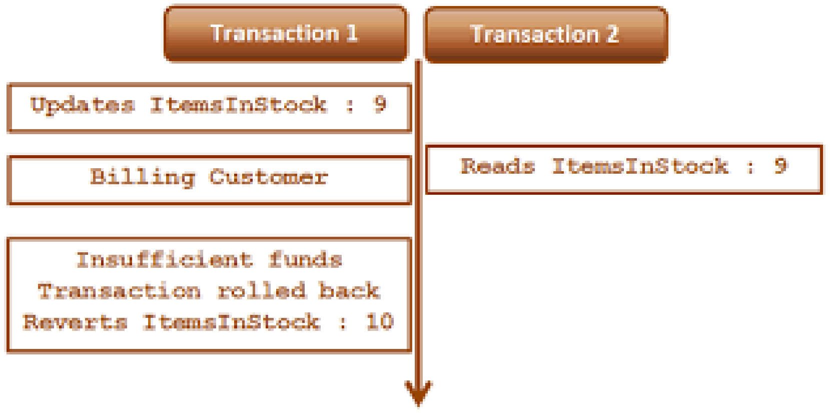
Transactions & Isolation Modes

	Dirty read	Nonrepeatable read	Phantom
Read Uncommitted	Yes	Yes	Yes
Read Committed	No	Yes	Yes
Repeatable Read	No	No	Yes
Serializable	No	No	No

cegeka

Read Uncommitted

- Reads all uncommitted data in its current form
- Allows dirty reads, which means it can read rows that have been modified by other transactions but not yet committed.
- It takes no locks and ignores locks from other transactions.



Read Committed

Allows reads on committed data only, in other words, it can't read data that has been modified by other transactions but not yet committed.

But it doesn't guarantee that rows read will stay consistent throughout the entirety of the transaction.

The Database Engine acquires shared locks as data is read and releases those locks when the read operation is completed.

Repeatable Read

Specifies that statements cannot read data that has been:

- Modified but not yet committed by other transactions
- No other transactions can modify data that has been read by the current transaction

,until the current transaction completes.

Shared locks are placed on all data read by each statement in the transaction and are held until the transaction completes.

Serializable

- Statements cannot read data that has been modified but not yet committed by other transactions.
- No other transactions can modify data that has been read by the current transaction until the current transaction completes.
- Other transactions cannot insert new rows with key values that would fall in the range of keys read by any statements in the current transaction until the current transaction completes.

Isolation Levels

- Read uncommitted
  - As the name implies, is that one transaction can read the data of another uncommitted transaction.
- Read committed
  - Also as the name implies, is that a transaction cannot read data until another transaction is committed.
- Repeatable read
  - When starting to read data (transaction is opened), modification operations are no longer allowed.
- Serializable serialization
  - The highest transaction isolation level. Under this level, transactions are serialized and executed sequentially, which can avoid dirty read, non-repeatable read, and phantom read. However, this transaction isolation level is inefficient and consumes database performance, so it is rarely used.

Table hints

- Table hints are used to override the default behavior of the query optimizer during the data manipulation language (DML) statement.

Table hint: Update lock

- Add "I intend to update this row/page/table"
- Implicitly sets the isolation level to REPEATABLE READ
- SQL Server does takes out an UPDATE lock without you have to ask for it
- You can explicitly ask for UPDATE LOCK

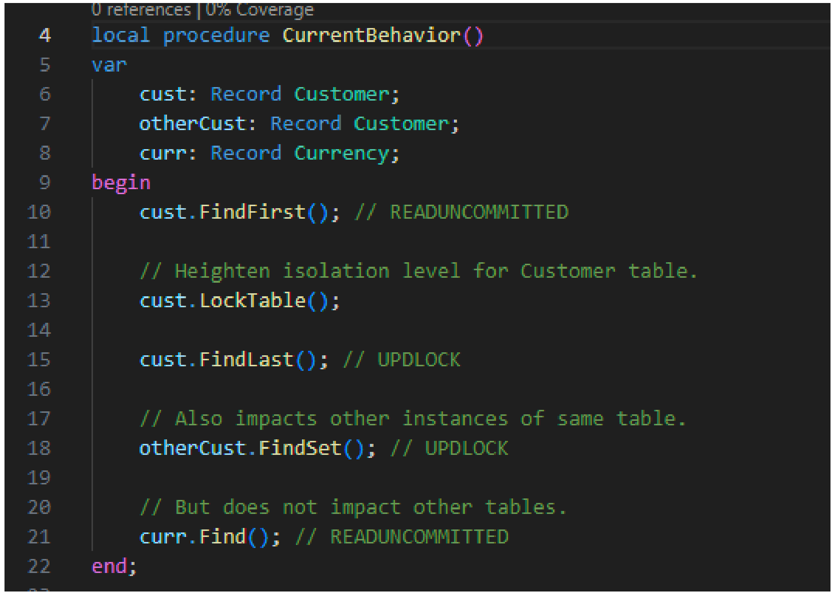
cegeka

Changing Isolation

BC 22 (W1 2023)

Default behavior

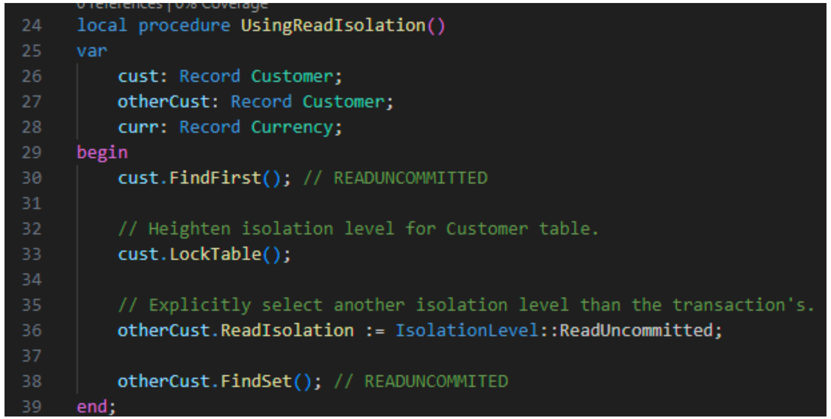
- Runtime determines the isolation level
- Isolation is heightened when:
  - Implicitly, trough record write
  - Explicitly, trough **LockTable**
- Heightened isolation remains the transaction for that table (not var)



```
0 references | 0% Coverage
local procedure CurrentBehavior()
var
    cust: Record Customer;
    otherCust: Record Customer;
    curr: Record Currency;
begin
    cust.FindFirst(); // READUNCOMMITTED

    // Heighten isolation level for Customer table.
    cust.LockTable();

    cust.FindLast(); // UPDLOCK

    // Also impacts other instances of same table.
    otherCust.FindSet(); // UPDLOCK

    // But does not impact other tables.
    curr.Find(); // READUNCOMMITTED
end;
```

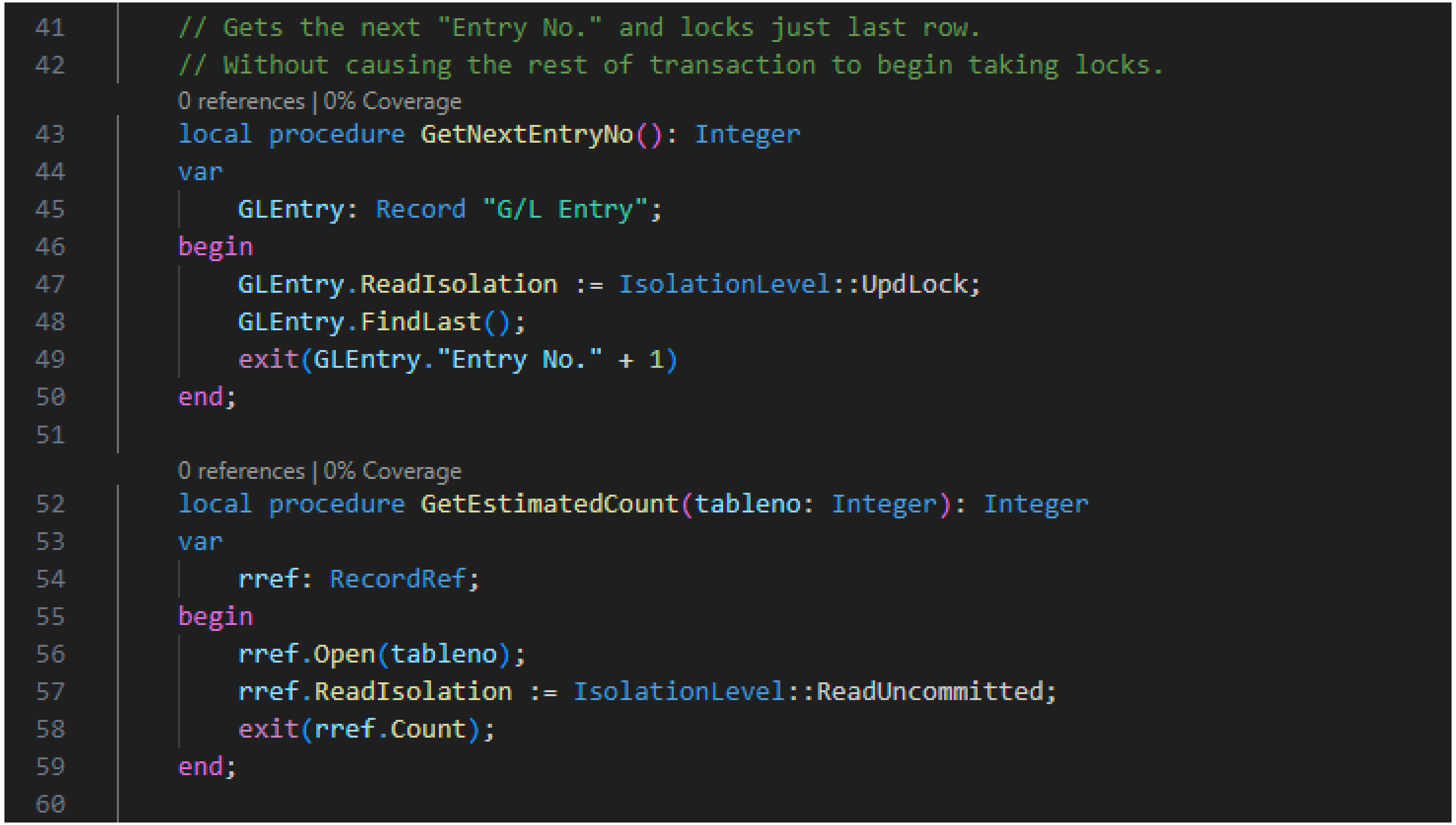
Using isolation level

- Explicitly, overwrite the default
- Heightening or lowering
- Set isolation remains the transaction for that table



```
local procedure UsingReadIsolation()
var
    cust: Record Customer;
    otherCust: Record Customer;
    curr: Record Currency;
begin
    cust.FindFirst(); // READUNCOMMITTED

    // Heighten isolation level for Customer table.
    cust.LockTable();

    // Explicitly select another isolation level than the transaction's.
    otherCust.ReadIsolation := IsolationLevel::ReadUncommitted;

    otherCust.FindSet(); // READUNCOMMITED
end;
```

Temporarily changing the isolation level



```
// Gets the next "Entry No." and locks just last row.
// Without causing the rest of transaction to begin taking locks.
0 references | 0% Coverage
local procedure GetNextEntryNo(): Integer
var
    GLEntry: Record "G/L Entry";
begin
    GLEntry.ReadIsolation := IsolationLevel::UpdLock;
    GLEntry.FindLast();
    exit(GLEntry."Entry No." + 1)
end;

0 references | 0% Coverage
local procedure GetEstimatedCount(tableno: Integer): Integer
var
    rref: RecordRef;
begin
    rref.Open(tableno);
    rref.ReadIsolation := IsolationLevel::ReadUncommitted;
    exit(rref.Count);
end;
```

cegeka

Demo

Isolation Levels

Tri State Locking

Optimized Locking

cegeka

Tri-State Locking

BC23 (W2 2023)

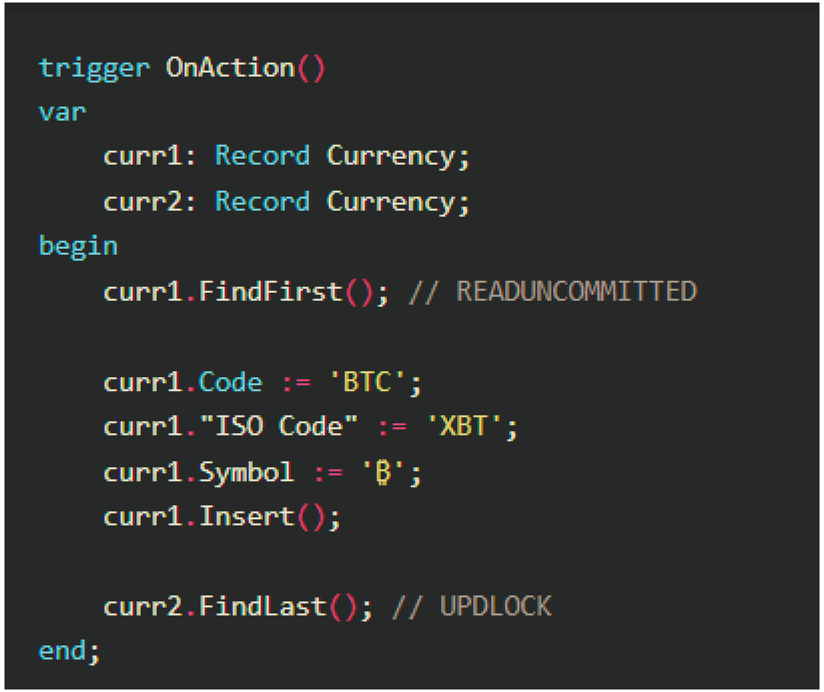
Locking Conflicts

Requested lock type	NO LOCK	(S)hared	(U)pdate	E(X)clusive
NO LOCK	No	No	No	No
(S)hared	No	No	No	Yes
(U)pdate	No	No	Yes	Yes
E(X)clusive	No	Yes	Yes	Yes

Simplified compatibility matrix of locks in SQL Server*. With No signifying no conflict between two requests and Yes as a conflict, leading the latter requested to have to wait.

Two-state locking

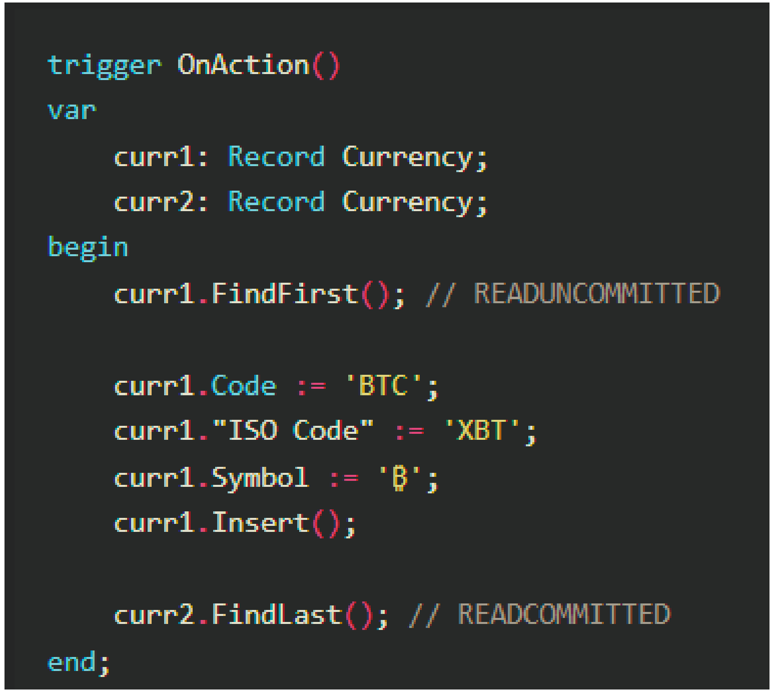
- All reads have the READUNCOMMITTED hint applied
  - As long as no writes have been done to the table in the current transaction
  - Nor LockTable has been called on a record of the table type.
- If writes have been done against the table (Or LockTable called), Further reads will have the UPDLOCK hint applied



```
trigger OnAction()
var
    curr1: Record Currency;
    curr2: Record Currency;
begin
    curr1.FindFirst(); // READUNCOMMITTED

    curr1.Code := 'BTC';
    curr1."ISO Code" := 'XBT';
    curr1.Symbol := '₿';
    curr1.Insert();

    curr2.FindLast(); // UPDLOCK
end;
```

Tri-state locking

- All reads have the READUNCOMMITTED hint applied
  - As long as no writes have been done to the table in the current transaction
  - Nor LockTable has been called on a record of the table type
- If writes have been done against the table in the current transaction, further reads will have the READCOMMITTED hint applied.
- If LockTable has been called on a record of the tables type in the current transaction, further reads will have the UPDLOCK hint applied.



```
trigger OnAction()
var
    curr1: Record Currency;
    curr2: Record Currency;
begin
    curr1.FindFirst(); // READUNCOMMITTED

    curr1.Code := 'BTC';
    curr1."ISO Code" := 'XBT';
    curr1.Symbol := '₿';
    curr1.Insert();

    curr2.FindLast(); // READCOMMITTED
end;
```

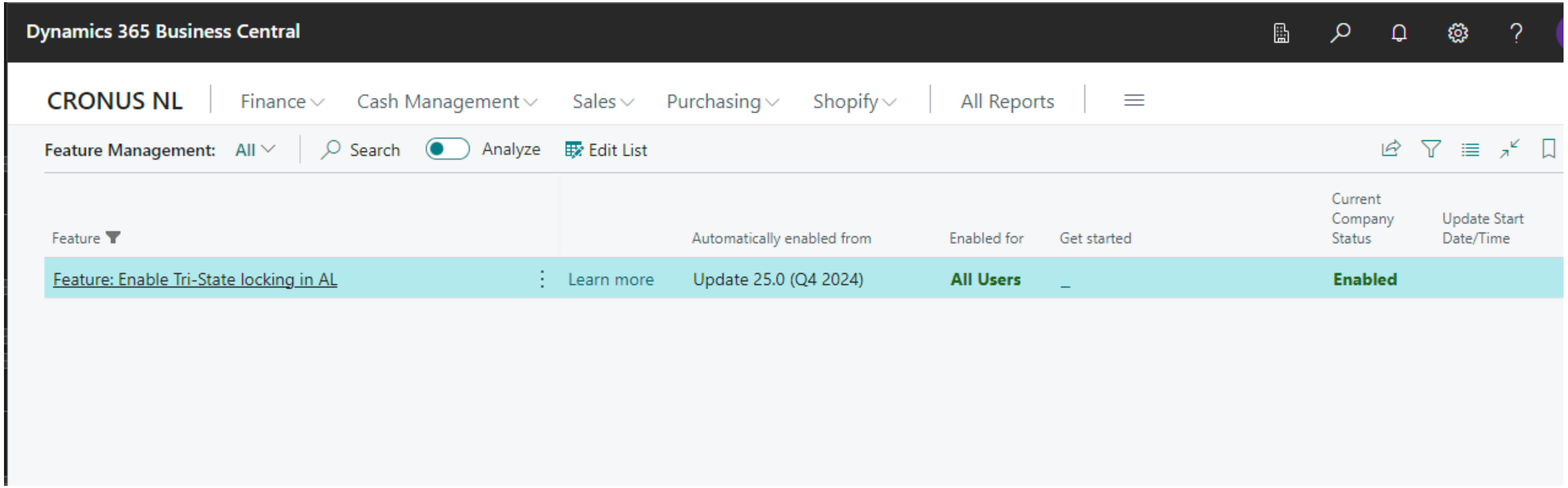
cegeka

Tri state locking

Properties	Locking behavior in versions 22 (and earlier)	Locking behavior with tri-state locking
Default isolation level for subsequent operations	UpdLock	ReadCommitted
Locking behavior	Session would acquire update lock on data from the table until it committed or rolled back its changes.	Session will only acquire a shared lock when reading data.
Consequences	Could cause blocking and contention issues when multiple sessions tried to access or modify the same table.	Allows other sessions to read and write to the same table concurrently, if they don't conflict with each other's changes.

cegeka

Tri-State locking



Feature		Automatically enabled from	Enabled for	Get started	Current Company Status	Update Start Date/Time
Feature: Enable Tri-State locking in AL	Learn more	Update 25.0 (Q4 2024)	All Users	_	Enabled	

Enabled by default (new tenants?)

Isolation Levels

Tri State Locking

Optimized Locking

cegeka

Optimized Locking

TBD

Locking levels

- Row Lock

  A row lock is the lowest level of granularity of locking possible in SQL Server. This means one or more specific rows will be locked, and the adjacent rows are still available for locking by concurrent queries.

- Page Lock

  A page lock in SQL Server will lock 8K worth of data even when your query only needs 10 bytes from the page. So your query will lock additional data which you do not request in your query.

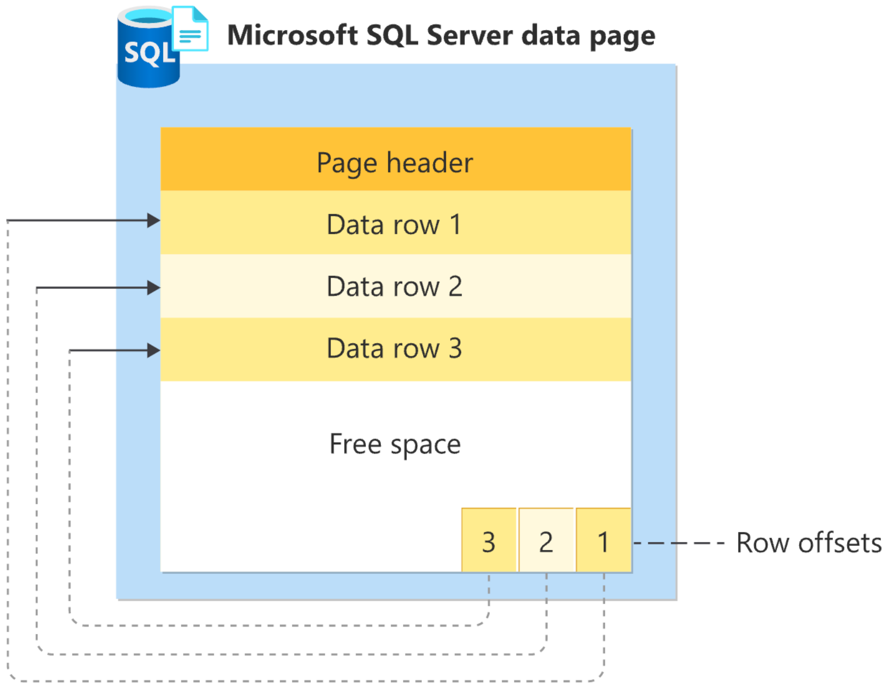
- Key Lock

  A keylock affects all rows that match the given predicate

- Table Lock

  A table lock will lock the complete table.

SQL Page


Microsoft SQL Server data page
Page header
Data row 1
Data row 2
Data row 3
Free space
3
2
1
Row offsets

Optimized Locking

Optimized locking is composed of two primary components: Transaction ID (**TID**) locking and lock after qualification (**LAQ**).

- **TID** is a unique identifier of a transaction. Each row is labeled with the last TID that modified it. Instead of potentially many key or row identifier locks, a single lock on the TID is used.
- **LAQ** is an optimization that evaluates predicates of a query on the latest committed version of the row without acquiring a lock, thus improving concurrency.

Optimized Locking

- Default has to be Read Committed
- Only Azure SQL (Certain regions)
- BC will adopt in the future

Resources

- SQL Hints
- SQL Transaction level
- BC Optimized Locking
- BC Tri-State Locking
- Dirty read, Non-repeatable read, and Phantom read

Thank you

cegeka