PUG Challenge EMEA

2018

Record Locking & Transaction Scope

What You Need to Know

Presented by: Dan Foreman

Dan Foreman CV

- Progress User since 1984
- Has anyone in the audience used Progress that long?
- Author of several Progress related Publications
  - *Progress Performance Tuning Guide*
  - *Progress Database Administration Guide*
  - *Progress System Tables*
- Author of several Progress DBA Tools
  - ProMonitor – used by Progress DBA Managed Services
  - Pro Dump&Load – dump & load with very short outage
  - Balanced Benchmark
- Getting closer to retirement!

BRAVEPOINT Progress White Star Software

Dan Foreman Miscellaneous

- Interests
  - Basketball
  - Cycling (my recumbent trike is made in the UK)
  - Backcountry skiing, also called randonee or ski mountaineering
  - Japan (**日本**)
  - World Travel
- Normally I share some pictures, take demographic surveys, etc. at this point but this is a long presentation so I need to get moving
- Please save your questions until the end

Note

- The code examples in this presentation are character style (i.e. non-OO) because it makes it easier to use a larger font and fit the example on one slide
- Dynamic Queries are not used for the same reason but the principles are the same
- I use the term 4GL not ABL
- 4GL Keywords are shown in upper case

Record Scope

- Every Table is ‘Scoped to’, or ‘associated with’, or ‘connected to’ a Block
- Which Block? The Block with the Record Scoping Property that is the outermost block containing references to the Table

Record Scope

- Blocks with the Scoping Property
  - FOR
  - REPEAT
  - Procedure
  - Trigger
- The DO Block does not have the Scoping Property unless it is added explicitly

Record Scope

- There are two kinds of Record Scope:
  - Strong Scope
  - Weak Scope
- We will discuss each of these shortly

Record Scope

- Record Scope can be verified with the COMPILE/LISTING option

```
File Name              Line Blk. Type Tran Blk. Label
-------------------- ---- --------- ---- --------------------
.\inquiry.p             0 Procedure No
    Frames:  MENU1

.\inquiry.p            26 Repeat    No   SUPER-BLOCK
    Buffers: cust.Customer
    Frames:  SERCH

.\cusfind1.i            8 Repeat    No   SEARCH-BLK
.\cusfind1.i           10 Repeat    No   SERCH-CTL
    Buffers: cust.Alt-Billing
```

Record Scope

- Why do we care about Record Scope?
- Record Scope can affect the duration of Record Locks
- The duration of Record Locks affects your job security
- Record Scope determines the maximum length of time a Record might be present in the Client's Local Buffer, but Record Scope does not guarantee that the Record is always AVAILABLE (4GL function)

Strong Scope

- Explicitly Scoping a Table to a Block using the FOR Option
- Example

```
/* r-scope3.p */
DO FOR customer :
    FIND FIRST customer.
END.
DISPLAY AVAILABLE customer.
/* Compile Error! */
```

Weak Scope

- This type of Scope is called “Weak” because references to a Table outside of a Weakly Scoped Block will raise the Record Scope to the next highest Block level
- Weakly Scoped Blocks
  - REPEAT Block
  - FOR Block
- Note that Trigger and Internal Procedure Blocks do not have the Record Scoping Property
- Example on next slide

Example

```
/* r-scope1.p */
FOR EACH customer :
        DISPLAY customer.
END.
DISPLAY AVAILABLE customer.
```

Definition of a Progress TRANSACTION

- A unit of work that must be done entirely or not at all
- The boundary of that unit of work is a 4GL block

Transaction Scope

- Every Transaction is Scoped to a Block
- Which Block? The outermost Block (or Peer Level Blocks) that contain changes to database tables
- Every Iteration of a looping Transaction Block is a New Transaction
- Verify Transaction Scope with the COMPILE/LISTING Option
  - But sometimes the LISTING Option might not be correct; Example to follow

Transaction Scope Example

```
/* trx-scope1.p */
FOR EACH customer :
        UPDATE customer.
        FOR EACH order OF customer:
            UPDATE order.
        END.
END.
```

Transaction Scope Example

```
/* trx-scope2.p */
FOR EACH customer :
        DO TRANSACTION :
            UPDATE name.
        END.
        FOR EACH order OF customer :
            UPDATE order.
        END.
END.
```

Transaction Scope

- Sub-procedures called from inside a Transaction Block do not start New Transactions
- The COMPILE/LISTING Option will NOT see this condition
- Only one Transaction can be active for a Progress Client; so if there are 20 Client connections to a Database, there can be a Maximum of 20 Active Transactions

Sub-Transactions

- Exist inside an Active Transaction
- Multiple, simultaneous (i.e. concurrent) Sub-Transactions are possible
- Purpose: to allow UNDO or error handling of smaller units within an Active Transaction

Example #1

```
/* caller.p */
RUN trx-debug.p PERSISTENT.
REPEAT WITH 1 DOWN 1 COLUMN :
    PROMPT-FOR order.order-num.
    FIND order USING order-num.
    UPDATE order-date ship-date promise-date.
    RUN called.p  ( BUFFER order ).
END.
```

Example #2

```
/* called.p */
DEF PARAMETER BUFFER order FOR order.
DO TRANSACTION :
    FOR EACH order-line OF order :
        UPDATE order-line EXCEPT order-num.
    END.
END.
```

Example #3

```
/* trx-debug.p */
ON F12 ANYWHERE DO:
   DEF VAR trx-status AS LOG FORMAT "Active/Inactive" NO-UNDO.
   trx-status = TRANSACTION.
   MESSAGE
         "Transaction Status:"    TRANSACTION    SKIP
         "Program Name:"          PROGRAM-NAME(2) SKIP
         “DBTASKID:”              DBTASKID(“dictdb”)
   VIEW-AS ALERT-BOX.
END.
```

Override the Default Transaction

- In this example the code is simple but
  - Performance is very slow because each item record requires a transaction start & a transaction end
  - Cannot be rerun in case of a problem

```
/* trx1.p */
FOR EACH item EXCLUSIVE-LOCK :
    item.price = item.price * 1.05.
END.
```

Override the Default Transaction

- This example can be rerun and is faster but Before Image file size and Record Locking could be a problem

```
/* trx2.p */
DO TRANSACTION:
    FOR EACH item EXCLUSIVE-LOCK.
        item.price = item.price * 1.05.
    END.
END.
```

Override the Default Transaction

- OK Dan, is there a better solution?
- Unfortunately the better solution is not always an elegant solution
- Try trx3.p (next slide)

```
trx-blk: REPEAT TRANSACTION:  /* item is scoped to this block */
    FOR EACH item EXCLUSIVE WHERE item-num GE last#
        v-counter = 1 TO 100 :
        price = price * 1.05.
    END.
    IF AVAILABLE item THEN DO:
        last# = item.item-num.
        FIND FIRST syscontrol EXCLUSIVE.
        syscontrol.last# = item.item-num.
        NEXT trx-blk.
    END.
    ELSE DO:  /* no more records to process */
        FIND FIRST syscontrol EXCLUSIVE.
        syscontrol.last# = 0.   /* Reset for next time */
        LEAVE trx-blk.
    END.
END. /* trx-blk: TRANSACTION */
```

Record Lock Types

- SHARE-LOCK
- EXCLUSIVE-LOCK
- NO-LOCK
- Record Get Lock (not covered in this presentation because a developer does not need to know what they are or what they do)
- Note that these are 4GL locks and not related to SQL Isolation Levels

SHARE-LOCK

- Default Lock mode for FIND, FOR, etc.
- Multiple Clients can Read the same record Concurrently
- A 4GL process can’t update a record with SHARE-LOCK Status; the Lock must be promoted (i.e. upgraded) to EXCLUSIVE-LOCK

EXCLUSIVE-LOCK

- SHARE-LOCKs are Automatically upgraded to EXCLUSIVE-LOCKs when the record is changed
- If a Client has an EXCLUSIVE-LOCK, no other Clients can Read the record unless NO-LOCK is used

NO-LOCK

- Can read any record even if another Client has an EXCLUSIVE-LOCK on the Record
- For this reason NO-LOCK reads are called a ‘Dirty’ reads because the state of the record cannot be guaranteed
- For Dynamic Queries the QUERY-PREPARE & GET-* methods are NO-LOCK by default

Default Locking Rules

- FIND, FOR, OPEN QUERY statements attempt to acquire a SHARE-LOCK on a Record
- When a Record is Updated, Progress automatically attempts to upgrade the lock to EXCLUSIVE-LOCK
- Default Lock for a DEFINE BROWSE is NO-LOCK which overrides the Lock specified by the associated OPEN QUERY Statement (bug?)

SHARE-LOCK Duration

- The end of Transaction Scope or the end of Record Scope whichever is later (i.e. ‘later’ at the Block level)
- Example to follow soon
- SHARE-LOCKs are usually Evil!

EXCLUSIVE-LOCK Duration

- The end of the Transaction!
- There are no exceptions to this rule no matter how hard you try

Downgraded Locks

- If the Record Scope is at a higher Block level than the Transaction Scope, an EXCLUSIVE-LOCK is Automatically Downgraded to SHARE-LOCK
- I call this a Limbo Lock because the Transaction has ended but the record is still locked; Progress has a different definition of Limbo Lock thatwe will see shortly
- Example on next slide

Downgraded Locks

```
Record Scope  Block
        Read Record
        Transaction Scope Block
            Change Record
        End
        Limbo Lock from here to end of Record Scope
        More code
End
```

```
/* serial2a.p - BROKEN VERSION */

REPEAT:
    DO TRANSACTION:
        FIND FIRST syscontrol EXCLUSIVE.
        syscontrol.last# = syscontrol.last# + 1.
        DISPLAY syscontrol.last# @ order.order-num.
    END.
    DO TRANSACTION:
        CREATE order.
        ASSIGN order.order-num.
        SET order-date promise-date ship-date.
    END.
END. /* repeat */
```

Fix the Program

- Strong Scope
  - The most efficient way
  - serial2b.p
- Re-FIND the record again with NO-LOCK
- RELEASE
  - Can be used in Two possible locations in the Program but let's talk about the RELEASE Statement first

RELEASE Statement

- RELEASE **cannot** Release an EXCLUSIVE-LOCK!!!!
- RELEASE can Release a SHARE-LOCK
- RELEASE **cannot** Release a SHARE-LOCK inside of a Transaction (undocumented)

RELEASE Statement

- If RELEASE is used inside a Transaction block, the Record Lock is flagged to be released when the Transaction ends
- RELEASE Flushes the Record from the Local Buffer
  - AVAILABLE = No
  - But this does not mean the Lock is necessarily Released

RELEASE Statement

- The Progress definition of Limbo Lock is a record that has been released but the Transaction has not ended
- This Limbo Lock shows a “L” in the Flags section of *promon* and the _Lock Virtual System Table

4GL Locking Options for Optimistic Locking

- The following statements will re-read the Record in the Buffer with the Specified Lock Status
- GET *CURRENT* <table>
- FIND *CURRENT* <table> statement
- FIND-*CURRENT* <table> method

4GL Locking Options for Optimistic Locking

- CURRENT-CHANGED <table>
  - Function
  - Attribute
- Returns *True* if the Record Retrieved with one of the previous CURRENT options is different from the Record in the Buffer before executing the CURRENT Record Retrieval

4GL Locking Options for Optimistic Locking

- NO-WAIT option
  - Option on DEFINE BROWSE, GET, FIND, CAN-FIND, FIND-BY-ROWID, and more
  - Can’t be used on FOR EACH
  - Doesn’t apply to TEMP-TABLEs
  - Raises the ERROR Condition
  - Behavior is different on a Browse

4GL Locking Options for Optimistic Locking

- LOCKED <table>
  - Logical Function that tells if the record is locked with SHARE or EXCLUSIVE status
  - Usually follows a NO-WAIT
  - See lock2.p
  - See lock4.p for even more options

Optimistic Record Locking

- A 4GL Program reads a Record using NO-LOCK, changes the data in TEMP-TABLEs, etc. and then re-reads the record EXCLUSIVE-LOCK to apply the changes to the Database
- It's 'optimistic' because we assume that usually no other process will touch the record in between the NO-LOCK and the EXCLUSIVE-LOCK - BUT YOU STILL NEED TO CHECK FOR CHANGES!
- Examples on next slide

Record Locking Examples

- lock1a.p (Character)
- lock1b.p (GUI)
- getcurnt.p (GUI)

Read Only (-RO) Option

- Client Startup Option
- No Record Locking of any kind
- Not even Record Get Locks
- No entry in the Database Log (.lg) File (until V10+)
- Not advised for use with a Production Database because the data returned to the Client might be logically corrupt
- Makes 4GL and Binary Dumps Faster (less so in V10 & V11)

-rereadnolock Startup Option

- Client Startup Option
- Introduced in V8.3B
- Critical for WebSpeed Agents and AppServers
- See next slide for an example

-rereadnolock Startup Option

- On AS1 User #1 Reads a Record NO-LOCK
- On AS2 User #2 Reads the same record and updates it
- On AS1 User #3 Reads the same Record NO-LOCK but sees the same copy that User #1 retrieved, not the updated version
- -rereadnolock Forces Progress to use the Newest Version of the Record

-rereadnolock Startup Option

- For more details see the Progress Kbase:
  - P43776
  - 19063
  - P12159

-lkwtmo Startup Option

- Lock Wait Timeout
- Client Startup Option (AppServer & WebSpeed also)
- Amount of time a Client will wait for a Record Lock before STOP Action occurs
- Timeout only occurs if the *NO-WAIT option is not used*, because a NO-WAIT will raise an ERROR, not STOP
- Default is 1800 seconds (30 minutes)
- Default for WebSpeed is 10 seconds
- Minimum value is 10 seconds (in older versions a value of zero meant wait forever)
- No screen or log (.lg) messages until V9.1D SP08

Monitoring Locks

- *promon*
- _Lock Virtual System Table
  - Be careful! Reading this table can cause performance problems
  - FOR EACH _Lock WHILE _lock-recid NE ?
  - Demonstrate deadly[1-2].p
- _UserLock Virtual System Table
  - Can see a Maximum of 512 Concurrent Locks
  - Prior to V10.1B did not have the Table#

The Cost of a Large Transaction

- Potential Record Lock Table Overflow
  - Approx 32 bytes per –L entry (V9 and later)
  - Finite Limit

The Cost of a Large Transaction

- BI File Size
  - 2gb maximum size prior to V9
  - Extents are limited to 2gb in V9/V10 unless the *proutil EnableLargeFiles* option is used
  - Crash Recovery will cause the BI file to **grow**
  - Why? Crash Recovery Notes for the DB changes caused by Crash Recovery
  - Limit BI Size with –bithold
    - Guideline: set no higher than 50% of Available BI Disk Space

The Cost of a Long Transaction

- Monitor Long Transactions with longtrx*.p
- Dan's Junior programming mistake….turning menu.p into a transaction

Conclusion

- Questions?
- Thank you for coming!
- Please don’t forget to thank the conference organizers for doing such a great job
- Dan Foreman
  - danf@prodb.com
  - +1 541 908 3437