

2-6 October, 2006

Hilton Vienna

Vienna, Austria

IDUG® 2006
Europe

E12

DB2 Application Design for Performance

Susan Lawson
YLA

Oct 4, 2006 • 16:15 – 17:15

Platform: z/OS

GoFurther





**Yevich, Lawson & Assoc. Inc.
2743 S. Veterans Pkwy PMB 226
Springfield, IL 62704**

**info@ylassoc.com
WEB: www.ylassoc.com**

IBM is a registered trademark of International Business Machines Corporation.
DB2 is a trademark of IBM Corp.

© Copyright 1998-2006, YL&A, All rights reserved.

Disclaimer PLEASE READ THE FOLLOWING NOTICE

- The information contained in this presentation is based on techniques, algorithms, and documentation published by the several authors and companies, and in addition is the result of research. It is therefore subject to change at any time without notice or warning.
- The information contained in this presentation has not been submitted to any formal tests or review and is distributed on an “As is” basis without any warranty, either expressed or implied.
- The use of this information or the implementation of any of these techniques is a client responsibility and depends on the client’s ability to evaluate and integrate them into the client’s operational environment.
- While each item may have been reviewed for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere.
- Clients attempting to adapt these techniques to their own environments do so at their own risks.
- Foils, handouts, and additional materials distributed as part of this presentation or seminar should be reviewed in their entirety.



Abstract and Outline

Today's application environments demand more availability and high performance than we ever had in the past. One problem with achieving both of these is locking. Locking can hold resources that affects availability and can slow down performance. We will take a look at some things we can do to minimize this without compromising data integrity. We will also look at application and database availability issues, because if you are not available....you are not performing!

- **Locks and Lock Avoidance**
 - Bind Parameters
 - Lock Avoidance
 - How it works
- **Application Design**
 - Designing for no deadlocks
 - Restructuring code for maximum throughput
 - Commit Strategies



Locks and Lock Avoidance

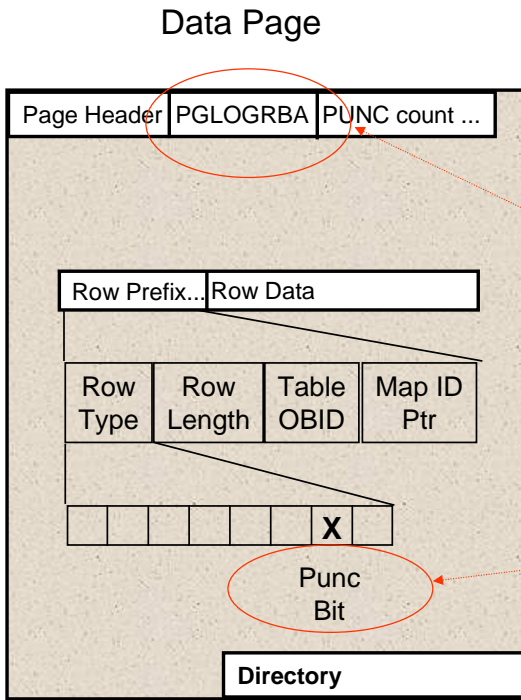


Lock Avoidance

- **Latches used in a way that deadlocks are avoided**
 - Latches do not time out
 - Latches are internal versus IRLM external locks
 - What about CS (cursor stability)?
 - If it is determined that the cursor
 - is read-only
 - appropriate options are set
 - the data is committed
 - then NO PAGE LOCK
- **Page latching function of the Buffer Manager**
 - Serialize access to a page
 - If required
 - Get page lock
 - Get row lock
- **Latches are physical only**
- **Latches are for pages only**



Lock Avoidance



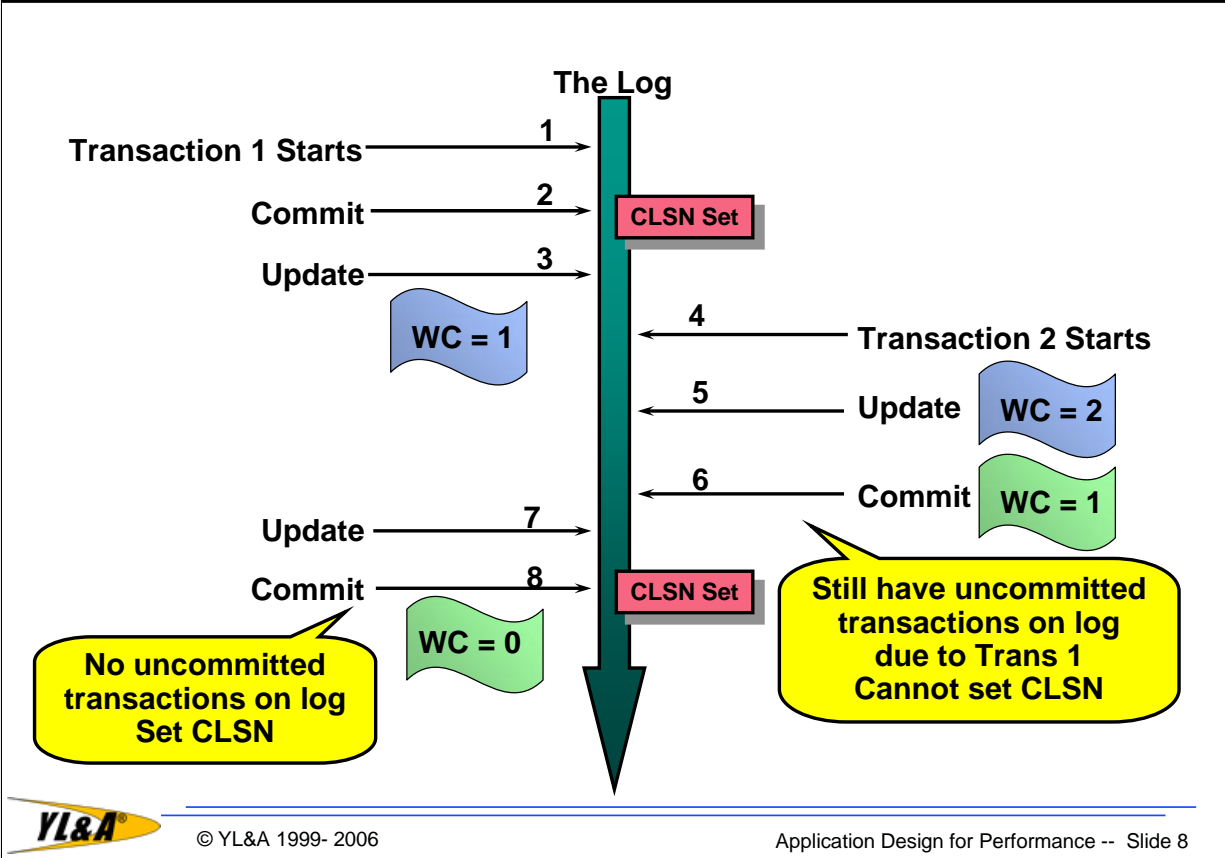
Rules for use:
CS
Read Only
CURRENTDATA(NO)

BM Latches Page

**Check the CLSN (> than PGLOGRBA)
NO then Check PUNC
PUNC Off
Return Row
Reject Row
PUNC On
LOCK**



Transactions Updating The Account Table



CURRENTDATA(NO)

- **Lock avoidance is critical for performance**
- **Only need CURRENTDATA (YES) if**
 - When the data returned to the application must not be changed before the next FETCH
 - Or, if you have SELECT followed by an UPDATE
 - Very rare that either is a requirement (but could be result of poor coding or 3rd party application)
- **Bind with CURRENTDATA(NO) and ISOLATION(CS)**
 - This must be specified as the default is CURRENTDATA(YES)
 - Applies to ambiguous cursors as well
 - This has been a problem in some applications
- **Lock avoidance also needs frequent commits**
 - PUNC bits can remain set
 - This requires locking
 - Frequent commits allow the CLSN to be updated more often
 - Still potential problem unless all applications do this
 - Data sharing uses a GCLSN
 - This is the lowest CLSN from all members
 - Longer commits cause long streams of updated pages
- **CHANGE ALL PROCESSES TO COMMIT FREQUENTLY**



Uncommitted Read

- **Make use of UR wherever possible**
 - Eliminates data locking entirely
 - Will almost never interfere with programs updating the database
 - If bound with ISOLATION(UR)
 - Mass deleters cannot run concurrently with UR program
 - Workfile TS cannot be dropped while a UR is using it
 - Staff must guarantee that UR can be tolerated in the processes
 - A good example is scrolling
 - Forward/backward looking for the right account
 - When found the update process will continue
 - Many potential problems also
 - If used should be at a single statement level
 - EXEC SQL SELECT ... WITH ISOLATION UR
 - Will still take CS lock for updates/deletes

Avoid Lock Avoidance



© YL&A 1999- 2006

Application Design for Performance -- Slide 10

RELEASE(DEALLOCATE)

- **One of the more misunderstood issues**
 - **RELEASE(DEALLOCATE) was always best**
 - **Thread reuse in CICS requires RELEASE(DEALLOCATE)**
 - **If you want the most CPU savings**
 - **RELEASE(COMMIT) problems with thread re-use**
 - **Each of these must be reacquired after COMMIT releases them**
 - **Tablespace locks**
 - **Some EDM pool control blocks**
 - **This is actually a long list (xPROCS, Inx Lookaside, sequential detection, etc)**
 - **Data Sharing impact**
 - **Tablespace/Partition L-locks get propagated to the CF**
 - **This occurs for just one member with interest in the TS**
 - **RELEASE(DEALLOCATE) holds onto the L-lock**
 - **RELEASE(COMMIT) causes**
 - **Release**
 - **Reacquire**
 - **Then re-propagate the L-lock**



RELEASE(DEALLOCATE)

- **RELEASE(DEALLOCATE) assists in reducing false contention**
 - A major portion of false contention is due to tablespace locks
 - XES-detected contention is primarily due to tablespace locks
 - XES System Lock Manager (SLM) recognizes only X and S locks
 - IRLM locks for tablespaces are usually IS or IX
 - RELEASE(DEALLOCATE) has nothing to do with data page X locks
 - This issue will be fixed in V8
 - These are released at COMMIT (except cursors WITH HOLD)
 - Tablespace locks are always INTENT locks except for 2 conditions
 - Application uses LOCK TABLE statements
 - Lock escalation occurs (and you are preventing this, right?)
 - Intent locks DO NOT INTERFERE with each other
- **Use for CICS thread re-use**
- **Use for some batch jobs, not all, but those w/many commits**
- **Be aware however...**
 - Remote connections will ignore this!
 - Movement of applications to Linux may see problems



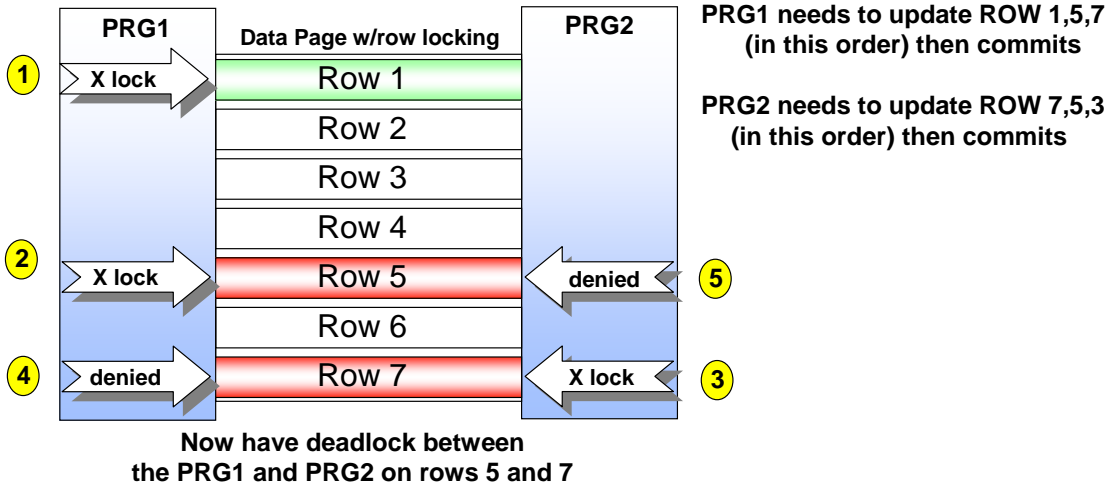
Row Level Locking

- **Is not a replacement for page locks**
 - Page locks will always be the most beneficial
 - Will allow much higher concurrency when required
 - Two users could be locking on the same page
 - Should ONLY be used when this is the business requirement
 - Ex: SAP – pushes large packages of updates/deletes to same data page constantly
- **Is not the answer to all concurrency problems**
- **Is not the answer to dead lock problems**
 - Can actually introduce more opportunities for deadlocks
- **Can be a great deal of overhead if used for high volumes**
- **If you must have the functionality consider..**
 - MAXROWS = 1
 - 1 row per page
 - Less chance for deadlocks
 - Better use for data sharing environments
 - Good when scanning control tables or look-up tables with an index



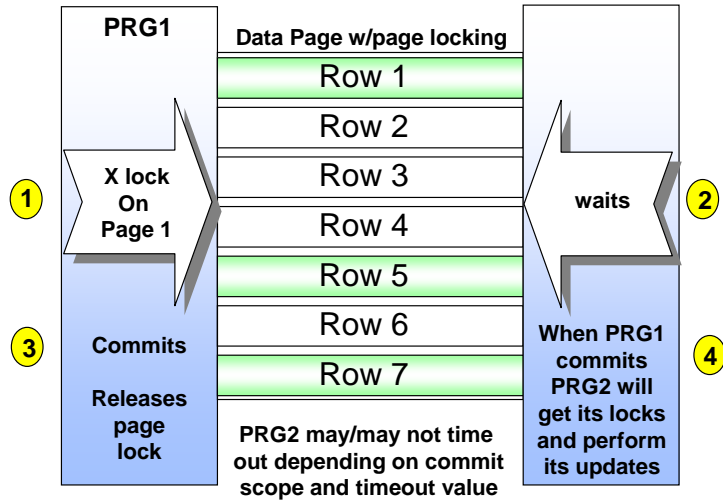
RLL Good for Concurrency ? – NOT ALWAYS !

- Ideally a more granular level of lock would provide better concurrency. But at what cost? Row level locking is more overhead and in some cases can cause more contention and possibilities for deadlocks.



RLL - Good for Concurrency Problems (continued)

PRG1 needs to update ROW 1,5,7 (in this order)
 PRG2 needs to update ROW 7,5,3 (in this order)



Row Level Locking Overhead

Elapsed Time and CPU Time for Locking Levels

LOCKSIZE	ELAPSED CLASS 2	CPU CLASS 2 LOCK	LOCK REQUESTS	MAX.LOCKS HELD
Table	13.37	1.98	2	25
Page	12.89	2.08	660	607
Row	21.68	3.27	23983	23826

Elapsed Time Executing DB2 Calls

Max Number of Locks Held by Program

CPU Time Executing DB2 Calls

**Row level locking almost doubled the CPU cost
And think about the extra memory needed to hold
all those locks!**

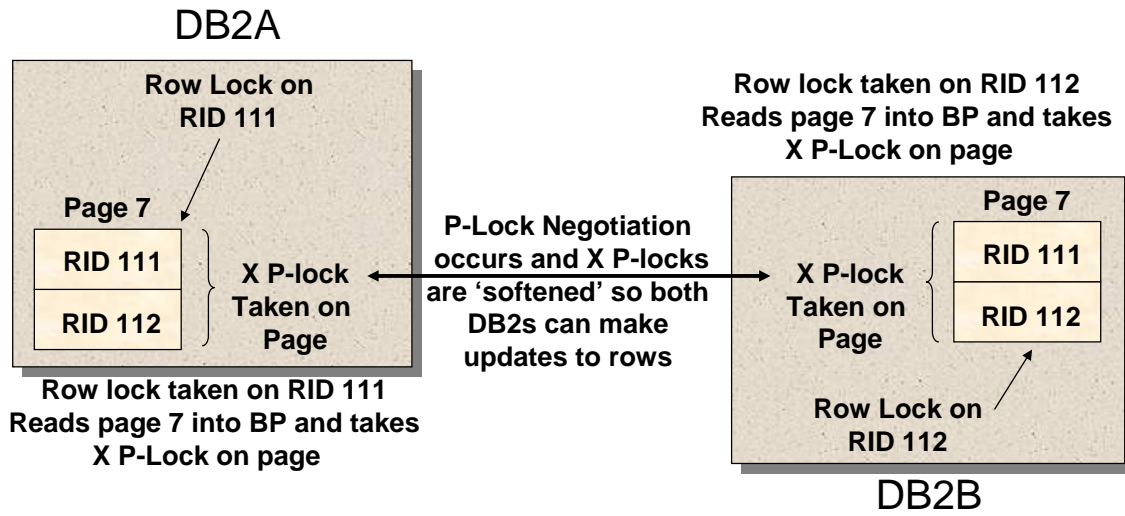
Source: IBM SG24-4725-00 Locking in a DB2 MVS Environment



© YL&A 1999- 2006

Application Design for Performance -- Slide 16

Page P-lock Used For Row Level Locking



Deadlocks, Timeouts and Retry Logic

■ Deadlocks and retry logic

- Insure RETRY logic is coded for applications requiring high availability
- For SQL used in applications requiring high concurrency, the SQL/process should possibly be retried if an SQL code of -911, -912, -913, -904 (data sharing) is received before abending the program [also with LOBS]
 - Five retries is a good starting point.

■ Timeouts vs. Lock Wait Time

- Myth:
 - ‘The application is performing well, we have only 1 or no *timeouts* per day’
- Reality
 - You may not hit the timeout (usually 60 seconds), but how much time IS spent *waiting* for a lock? 10 seconds? 25 seconds? 59 seconds?
 - Monitor the lock wait times from accounting and statistics reports
 - Some lower timeout to 5 seconds to immediately deal with problems



No Deadlocks, No Timeouts – No Problems? NOT ALWAYS!

- The fact that a particular problem does not occur does not mean there is not a performance issue/problem to be investigated. Often times this is due to focusing on reactive tuning measures (counting the number of deadlocks or timeouts), not on proactive tuning (monitoring lock waits)

CLASS 3 SUSP.	ELAPSED TIME	EVENTS
LOCK/LATCH	36.000000	64
SYNCHRON. I/O	1.4068000	539
DATABASE I/O	1.4068000	539
LOG WRITE	0.000000	0
. . .		

Accumulated Lock/Latch time

**0.5625
average
wait time**

Number of events

LOCKING	TOTAL
. . .	
LOCK REQUEST	50
UNLOCK REQUEST	24
. . .	
LOCK SUSP.	2

Total number of locks

**Do not
want these
to be close!**

Total number of waits on locks (ideally zero or close)



LOCKSIZE ANY and Lock Escalation

- You can still get lock escalation, even if you are defined with LOCKSIZE PAGE, ROW, or TABLE
- The determining factors are....
 - If you specify LOCKSIZE PAGE and omit the LOCKMAX during CREATE TABLESPACE the LOCKMAX defaults to 0 and escalation does not occur.
 - Only if you specify LOCKSIZE ANY the LOCKMAX defaults to SYSTEM.
 - On all other LOCKSIZEs the LOCKMAX defaults to 0 disallowing escalation!
 - If ALTERed from LOCKSIZE ANY (with LOCKMAX SYSTEM default) to LOCKSIZE PAGE, the default remains SYSTEM and escalation still occurs

CREATE TABLESPACE...LOCKSIZE ANY

Defaults to SYSTEM → lock escalation at NUMLKTS

CREATE TABLESPACE...LOCKSIZE PAGE

Defaults to 0 → Disallows lock escalation



LOCKSIZE ANY cont...

- This of course can be avoided by specifying LOCKMAX SYSTEM or LOCKMAX <value>
 - If you choose LOCKMAX SYSTEM, the NUMLKTS must be reached before lock escalation will occur
- In general most do not specify LOCKMAX and then the LOCKSIZE PAGE can be a heavy resource hog
- Caveat...
 - When allowing all tablespaces default to LOCKSIZE ANY
 - You are letting DB2 choose where to *start* locking
 - Based upon your process this could be page, table or tablespace
 - Do we want DB2 choosing? Maybe.....maybe not

Just a thought....

- Why not turn off lock escalation???
- Set LOCKMAX = 0
- Increase NUMTKTS appropriately, watch commits and mass processes
- It has been done before successfully!



Volatile Tables to Avoid Deadlocks for Clustered Data

■ Volatile tables (Version 8)

- Prefer index access
 - Over tablespace scans or non-matching index scans for tables that have statistics that make them appear to be small
- Good for tables that shrink and grow
- Allow matching index scans on tables that have grown larger without new RUNSTATS

```
CREATE TABLE NEW_TABLE
(COLA ...)
VOLATILE
```

*Will disable list prefetch and
some other optimization options*

```
ALTER TABLE NEW_TABLE
VOLATILE
```

■ Cluster tables are those tables that have groups or clusters of data that logically belong together

- Within each group rows need to be access in same sequence
- Often found in 3rd party applications (SAP)
- Can cause lock contention during concurrent access
- Sequence of access is determined by primary key and if DB2 changes a new access path lock contention can occur



Cluster Table - Concept

■ Exactly what is a cluster table??

- Table made up of logical rows
 - With each logical row consisting of multiple physical rows from that table
 - Logical row is identified by a primary key and a sequence number
 - Provides logical ordering of physical rows
- Accessing the logical rows are to be in the order of the primary key and the sequence number

Index: Jobcode, Sequence
JOBTABLE

Jobcode	Seq	Rank
DBA	1	Senior
DBA	2	Junior
DBA	3	Trainee
Analyst	1	Senior
Analyst	2	Junior
User	1	Senior

- Reduced deadlock chances by two different applications that access same logical row
 - But access the physical rows in a different order

Access to physical rows in cluster must be by the index to ensure they are access by Sequence to avoid deadlocks.

```
SELECT * FROM JOBTABLE
SELECT * FROM JOBTABLE WHERE JOBCODE = 'DBA'

NOT VOLATILE – Scan and IX1 with List Prefetch
VOLATILE – Both are IX1 with no prefetch
```



Dynamic Scrollable Cursors and Deadlocks

- Be careful with dynamic scrollable cursors in Version 8
- Rows can be accessed and updated in any order
 - Be sure to commit often!

FETCH FIRST
 FETCH ABSOLUTE +9
 FETCH RELATIVE -3
 UPDATE WHERE CURRENT OF

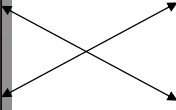
FETCH FIRST
 FETCH ABSOLUTE +9
 FETCH RELATIVE -5
 UPDATE WHERE CURRENT OF

FETCH RELATIVE -2
 UPDATE WHERE CURRENT OF

FETCH ABSOLUTE +6
 UPDATE WHERE CURRENT OF

TXNDATE		AMT
0925	1)	+200
1001		-150
1003		+500
1007	4)	-100
1008		-100
1010	3)	+50
1021		+300
1103		-150
1121	2)	+100

TXNDATE		AMT
0925	1)	+200
1001		-150
1003		+500
1007	3)	-100
1008		-100
1010	4)	+50
1021		+300
1103		-150
1121	2)	+100



A Comment on Locking...

**The best lock is
the one not taken**

- **We need to change the way we design applications**
 - Higher volumes and higher availability
- **What worked in the past may be hurting you today**
 - Especially if you are trying to meet more demands



Application Design



Batch Cursor and Restart Issues

■ **Watch out for CURSOR HOLD**

- CURSOR HOLD prevents DRAINS
 - Can stop On-line reorg
 - Can stop partition independence
- If used, make sure it is not for long periods
- Prevents CPU parallelism
 - Cursor's use is potentially interrupted by a commit
- Commit Strategies
- When to COMMIT → dynamic controls by updates and time
- Testing ROLLBACK → need to guarantee precisely
- Can also prevent dropping of Declared Temporary Tables in V8

■ **Checkpoint/Restart**

- How to standardize – never a 'one size fits all'!
- Is this really truly necessary
- How much is being spent in resources for that one 15 minute restart per year?
- Would rolling back actually be faster? Cheaper?



Identifying Long Running Units of Work

- In our high volume, high performance, high availability environments, it is imperative that commits be performed
 - Lock Avoidance
 - Concurrency
 - Restart
 - Rollback/Recovery
 - Utility processing
 - Resource release
- Can find these trouble applications, before they find you
- DSNZPARM URCKTH used for finding programs that are not committing
 - This parameter specifies the number of checkpoints that should occur before a message is issued identifying a long-running unit of work

URCHKTH

- Consistent checkpoint intervals will also enhance the identification of long-running units of work



Long Running Readers Warning – Version 8

- Proactive identify reader threads that have exceeded the user specified time limit threshold without COMMIT
- Writes IFCID 313
 - Each time the time interval is exceeded
- New DSNZPARM in Version 8
 - Long running reader threshold

LRDRTHLD

- When non-zero
 - DB2 records time a task holds a read claim
 - When passed specified number of minutes record is written
- Values
 - 0 – 1439 minutes, default 0 (off)
 - Online updateable
- Long running reader claims can prevent online reorgs
 - Need to ensure all applications are committing!



Interesting Rollback Dilemma

Lock and latch suspensions	10
Elapsed Time	.000045
. . .	
Maximum page or row locks held	3
. . .	

89K of Random updates over 60 Million pages with only 3 locks taken

Program expected to do approx. 1 insert/3 updates ... far below this ratio

Total DML	476,118
Select	14,628
Insert	2
Update	89,407
. . .	



Commit Strategy and DB2 Internal Checkpoints

- **The frequency of Commits**
 - Increases the number of log records written
 - Increases the number of log checkpoints
 - And vice versa
 - Must have a strategy for this!
- **Critical to logging**
 - Commit frequency!!!
 - Table design
 - Effective SQL
- **SQL?**
 - The program that updates unnecessary columns, etc.
 - Causing excessive logging and more frequent checkpoints
- **DSNZPARAM LOGLOAD and CHECKFREQ**
 - Determines how often DB2 checkpoints the log
 - SET LOGLOAD command or online ZPARAM (LOGLOAD or CHECKFREQ)



Heuristic Control Tables

- Rows unique to each application process used to control the commit scope using the number of database updates or time between commits
- Accessed every time an application starts a unit-of-recovery
 - Which would be process initiation or a commit point.
- Read from the table at the very beginning of the process to get the dynamic parameters and commit time to be used
 - Values in these tables can be changed either through SQL in a program or by a production control specialist
 - Able to dynamically account for the differences in processes through time
 - May change the commit scope of a job that is running during the on-line day vs. when it is running during the evening.

<i>Rows updated between commits</i>	JOB Name	Commit Frequency	Stop Indicator	Commit Counter	Last Commit	Restart Info	<i>Commits since start of job</i>
	JOB1	50	Y	400	12:10		
	JOB2	10	Y	100	02:00		
	JOB3	10		2000	14:19		<i>Time of last commit</i>



SAVEPOINT (External Savepoints)

- **External savepoints**
 - Enables milestones with a unit of recovery
 - Represents the state of the data and schema at a particular point in time
 - Can use ROLLBACK to restore to a savepoint
- **You have reached a point during a UNIT-OF-WORK**
 - Need to backout to without ROLLBACK over the entire UNIT-OF-WORK
- **Can name individual SAVEPOINTS**
 - ROLLBACK to which point is required based on requirements
 - Can leap over individual savepoints
- **Minimal Overhead**

```
Code...  
SAVEPOINT AAA  
SELECT ...  
Code...  
SAVEPOINT BBB  
Code...  
ROLLBACK TO SAVEPOINT AA
```



SAVEPOINTS – Rollback and Restrictions

- **To rollback to a savepoint use the ROLLBACK TO SAVEPOINT xxxx**
 - Backs out all data and schema changes made after the savepoint
 - The following actions will not be rolled back
 - Updates outside local DBMS (remote DB2s, VSAM, CICS, IMS)
 - Changes to global temp tables, however declared temp changes will be backed out
 - Opening/closing cursors
 - Cursor positioning
 - Locks acquired/released
 - Caching of rollback statements
- **Savepoints cannot be used with...**
 - Global transactions
 - Triggers
 - User-defined functions
 - Stored Procedures, UDFS or triggers nested within UDFs

**Can optionally use
ON ROLLBACK RETAIN CURSORS
ON ROLLBACK RETAIN LOCKS**



OLTP: SQL Transaction Cost

<p>Sign-On/Identify AUTHID is passed to DB2 Catalog access for authorization check (with catalog lock)</p> <p>Check Max Users 'WAITS" if at maximum number of threads (ZPARM)</p> <p>Thread Creation SKCT Directory and Header are loaded CT (Cursor Table) is created Directory Access for PLAN parts</p> <p>Check Plan Authorization Catalog access - check AUTHID (with catalog lock) Catalog access - objects AUTH checked (with catalog lock) Automatic rebind occurs here if required - (with catalog locks)</p>	<p>Resource Allocation Load DBDs Allocate and open all datasets (underlying VSAMs) Acquire locks - (tablespace, index space locks)</p> <p>Process SQL EDM process RDS process DM process BUFMR process Data access (Index access) Predicate processing Sorting</p> <p>COMMIT Processing Log buffers are written to DASD Locks are released</p> <p>Thread Termination Accounting records are written Storage is freed Objects are closed (CLOSE = YES [maybe])</p>
---	--



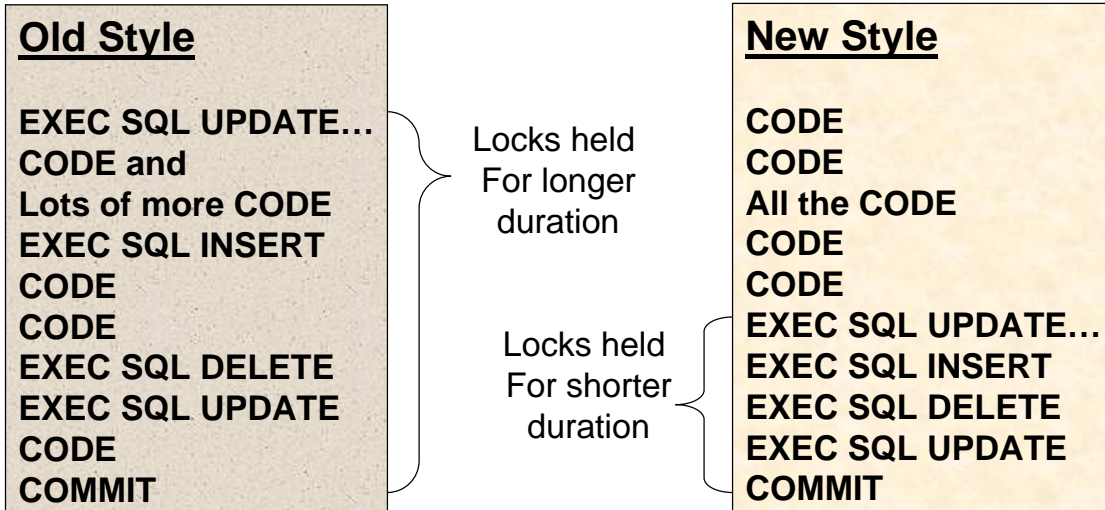
OLTP and Batch – Locking and Concurrency

- **Concurrency with batch**
 - Updates Real or Pseudo
 - In the online just flag the row for deletion
 - In the batch, do the real deletion
 - Needs very specific index to prevent scanning - often not there
 - Overhead of delete not critical unless RI cascading issues
 - This strategy needs to be proven and tested
- **Lock Acquisition**
 - Acquire(USE) Release(DEALLOCATE) (at Thread Term)
 - Thread Reuse
 - Keep all resources acquired and there are many besides intent locks
- **Transaction Design Issues**
 - What is a Transaction
 - Rules for Transaction Design
 - All non-SELECT SQL at end of commit scope
 - All non-SELECT objects in predefined order
 - Look at the diagram on the next page



Load Reduction By Application Transaction Re-design

- **Move all physical changes to end of transactions**
 - Not where they “logically” occur
- **Apply DML to tables in a pre-determined sequence**
 - To avoid lock contention and deadlocks



SQL Exploitation

- **Filter *as much as possible* and *as early as possible***
 - Most restrictive predicates first
 - High filter factor
- **Make repetitive processes efficient**
- **Code for Indexable Stage 1 predicates**
 - Then Stage 2
 - Don't filter in 'Stage 3' if at all possible
 - Promote those predicates!
- **Code predicates with a predetermined order**
- **Code subqueries with those that reject most, first**
 - Pick appropriate subquery type based upon available indexes
- **Be selective about the columns being returned**
 - It adds up!
- **Know your data (HAVE STATISTICS!!!!)**
 - Test and Prod.....is the data distribution representative?
- **Monitor, Tune and Test**



Summary and Conclusions

- **Two biggest challenges today for databases and applications:**
 - Concurrency and availability
- **Coding to solve a business problem is only the beginning**
- **High volumes and high availability dictate more advanced designs**
 - Code restructuring
 - Table restructuring
 - Transparent access during maintenance
- **Coding in a traditional manner may not work**
 - More care need for resource sharing and release
 - More care needed in design before coding begins
- **Designing high availability applications will also require databases designed for high availability**
 - And vice versa
 - One cannot accomplish this goal alone
- **No silver bullets for 24X7 availability and performance**
 - Must start in design
 - V8 does not have the answers (i.e. DPSIs)



Theory into Practice!!!

- **Large financial client**
 - 1.5 million input records
 - Qualified over 50 Million more
 - Narrowed down to 45K
 - Scaled linear from 1 – 100 threads
 - About 1 sec per thread, working its way down still
- **Locking**
 - 48 timeouts (timeout at 10 seconds for online...)
 - 1 deadlock (will find it!!!)
 - Why so low?
 - RESPECT for input (design for input access)
 - RESPECT for updates (all at end in sequence)

This stuff really works!!!

Performance and availability is NEVER included in traditional application and database designs!!!!



DB2 Information on the Web

- **IBM**
 - ibm.com
- **IBM Software**
 - ibm.com/software
- **DB2 Family**
 - ibm.com/software/db2
- **DB2 Solutions Directory Applications**
 - ibm.com/developerworks/db2
- **"Red Books"**
 - ibm.com/redbooks
- **DB2 for z/OS**
 - ibm.com/software/db2zos
- **DB2 Support**
 - ibm.com/software/db2zos/support.html
- **DB2 for z/OS Papers**
 - <ftp://ftp.software.ibm.com/software/data/db2zos>
- **DB2 Magazine**
 - <http://www.db2mag.com>
- **DB2 Certification**
 - <http://www.ibm.com/certify>
- **DB2 Experts**
 - www.db2expert.com



E12 DB2 Application Design for Performance

Susan Lawson
YLA
Susan_lawson@ylassoc.com



© YL&A 1999- 2006

Application Design for Performance -- Slide 42