Demystifying Logging &Recovery in DB2 for LUW

Dale McInnis (dmcinnis@ca.ibm.com)
STSM, Hybrid Data Management Technical Sales
IBM Canada Ltd.

For many DBAs, how DB2 captures the changes to a database and can use that to recover from failures is a

mystery. But with over 20% of the 100+ configurable database configuration parameters being related to logging and recovery, it’s important to understand how it all works and how you can be sure that you’re logging in an efficient matter and that you’re protected from various types of failures. In this presentation you'll learn about logging strategies, how to properly configure the most important parameters (for feature and performance), and how to monitor the logging that is happening in your environment. Topics also include logging basics and concepts, mirrored logging, infinite logging, log archiving, types of recovery (crash recovery vs. roll forward), minimizing crash recovery time, and methods for limiting what gets logged.

This presentation is valid as of DB2 11.1.1.1.

Agenda

- Logging Basics and Concepts
- Log Consumption
- Configuring the Active Log Space
- Monitoring Logging Activity

The presentation is broken up into three main sections that will cover the following

topics:

- Understand the basics and concepts associated with logging and recovery in DB2 LUW
- Determine how to configuring logging (for feature and performance)
- Describe the active log space and how to configure it
- Determine how to limit what gets logged and how to minimize crash recovery time
- Learn how to monitor logging activity for the database and individual transactions

Logging Basics

- **Transaction logs written by DB2 to record updates to database**
- **Used during**
  - Rollback
    - Reverse changes made by a statement or transaction
  - Crash recovery
    - Redo committed work and undo uncommitted work to make database consistent
  - Rollforward recovery
    - Re-apply changes after a restore is performed
- **DB2 uses a Write-Ahead-Logging (WAL) protocol**
  - Log records always written to disk before affected data pages hit disk

This section will cover logging basics – what it is and how it is used within DB2.

Logging Basics (cont.)

- **Each log record identified by a Log Sequence Number (LSN)**
  - Ordering and addressing scheme for log records
  - Assigned when log record is written
  - Page LSN field in data page header updated with the LSN of the log record associated with the most recent change to the page
    - During recovery, allows us to tell what changes have already been made to a page
    - Think of it like a version number for the page that represents the time of the last update to it
- **Log records are initially written into an in-memory log buffer but are then flushed to disk when one of the following things happens**
  - The log buffer is full
  - A transaction commits (must maintain durability in case of system failure)
  - Data pages are written to disk (to ensure uncommitted changes to the page can be undone after system failure)

These are internal details that might not be of interest to many DBAs but it's explained here for those that like to get deeper into the technology. And LSNs are often seen in the db2diag.log file so people do tend to want to understand what LSNs are. You'll also see them come into play later in this presentation.

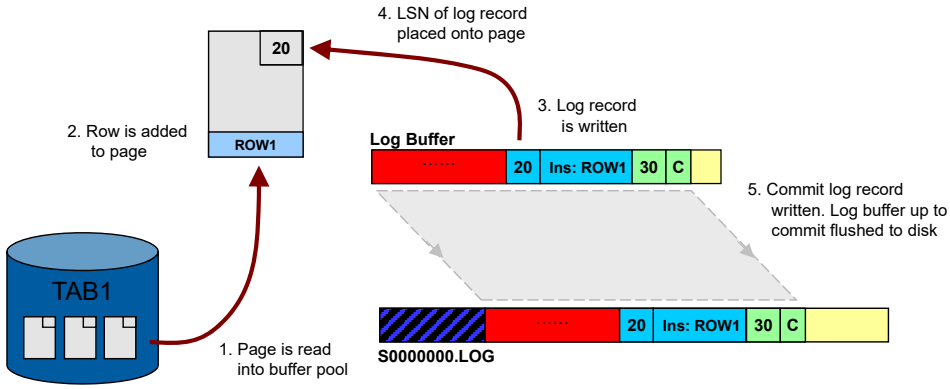
Logging Basics (cont.)

- **Log records are written into log files**
  - First two pages of each log file are reserved for metadata
  - Remaining pages are used to hold log records
  - # of 4KB pages in a log file is: `LOGFILSIZ + 2`
- **Log files are numbered starting with `S0000000.LOG`**
- **When log records are written by transactions, extra space is reserved in case a rollback occurs**
  - Undo (a.k.a. compensation) log records are written during rollback processing
  - May require as much space as corresponding redo log records but typically less
- **In DPF, each database partition has its own set of log files**
- **In pureScale, each member has its own set of log files**

Further logging basics.

Example of Writing Log Records

```
INSERT INTO TAB1 VALUES ('ROW1');  COMMIT;
```



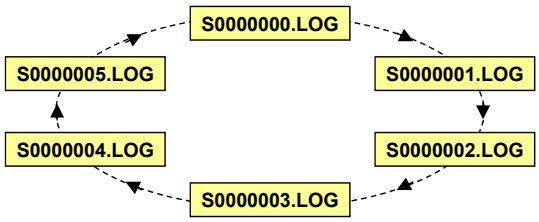
Note: Pre-9.8 log records do not contain the LSN. Done here for illustrative purposes.

In this example, an INSERT statement (into a row-organized table) is being

executed and is subsequently committed. This shows what happens within DB2 when this happens.

Recoverability – Circular Logging

- **Default for newly created databases**
- **Contents of log files are not permanent**
  - Log data is overwritten when log records are no longer needed for crash recovery purposes
- **Only offline database backups are permitted**
- **Rollforward not permitted**

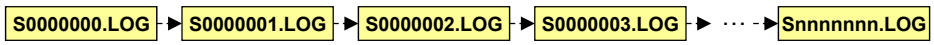


The default for newly created databases is circular logging. This is not typically

used in production environments but is sometimes used in test and development environments – as it makes log management easier.

Recoverability – Recoverable/Archived Logging

- **Enabled by setting `LOGRETAIN`, `USEREXIT`, or `LOGARCHMETH1/2` database configuration parameters**
  - `LOGRETAIN/USEREXIT` parameters no longer exist in DB2 10.1
    - Use `LOGARCHMETH1` instead
- **Log files are retained (never reused) and can be archived**
- **Permits the use of**
  - Online backup
  - Table space backup and restore
  - Database and table space rollforward
  - Incremental backup
  - Rebuild from table space backups



Most production environments use recoverable databases – because the benefits of

using it provide more flexibility and better recovery characteristics.

Recoverability – Recoverable/Archived Logging (cont.)

- **For archived logging, consider using automated log file management**
  - Choose how long to retain recovery objects like backups and archived logs and when to automatically prune them
  - See `NUM_DB_BACKUPS`, `REC_HIS_RETENTN`, and `AUTO_DEL_REC_OBJ` database configuration parameters
- **Log archival compression (added in DB2 10.1)**
  - Log files can be compressed when they are archived
    - Applicable to disk, TSM, and vendor archiving methods
  - Enabled via `LOGARCHCOMPR1/2` database configuration parameters
  - Even if data inside log records is compressed, still see good (2-3x) compression on the log files
  - Still looks like a log file, but DB2 knows its compressed

<No speaker notes for this slide>

KS-INTERNAL:

NUM_DB_BACKUPS

full database backups to retain

Older full backups and associated incremental backups, table space backups, and load copies are marked as expired

Log files entries are not included and are kept as-is

Next full backup will prune the entries

REC_HIS_RETENTN

days that historical information on backups will be retained

When backups are pruned, corresponding recovery objects (including logs) are pruned as well

Most recent full database backup plus its restore set will always be kept

AUTO_DEL_REC_OBJ (V9.5+)

Specifies whether recovery objects are deleted when their history entries are pruned

Pruning occurs after a successful full database backup (based on parameters above) or when PRUNE HISTORY is used

These are all database configuration parameters. Various rules dictate how they work together so best to read up on them in the Information Center.

The defaults are:

- AUTO_DEL_REC_OBJ: OFF
- NUM_DB_BACKUPS: 12
- REC_HIS_RETENTN: 366

Encryption & Log Files (DB2 10.5 FP5+)

- **Encrypted databases include encryption of both the active log files and the archived log files**
- **Log data encrypted as it is written to log files and decrypted when read from log files**
- **Encryption of log files tied to the encryption key of the database at the time the log file was generated – does not correspond to encryption key of backup**
- **If rollforward required on a DR system, the keystore on that system must include the key(s) associated with all of the archived log files**
  - Because if a key rotation is performed on the database, it does not impact log files already generated

Native Encryption was added into DB2 in DB2 10.5 FP5.

We encrypt the active logs if the database is encrypted. The archived logs are basically just copies of the active log files, we don't decrypt/encrypt as part of archiving.

At the time they are created, we don't know if the backup will be encrypted – since you can specify encryption options on the backup command itself.

If somebody in a DR scenario has to restore and rollforward then they would have to have both keys (one associated with backup image and one associated with archived log files) in their keystore on the target system. And if the master key had been rotated, then the previous values as well. We do not rotate the key for archived logs.

Logging EDUs

EDU Name	Description
`db2loggr`	Log reader; also responsible for other things (soft checkpoints, reclaiming log space, suspending logging, etc)
`db2loggw`	Log writer
`db2logts`	Tracks what table spaces have log records in what log files (so unnecessary log files can be skipped during table space rollforward)
`db2lfr`	Log file reader for processing individual log files; used as an LFR/Shredder pair
`db2shred`	Shredder; extracts log record from log pages received from the LFR EDU (one per scan)
`db2redom`	Processes redo log records during recovery and assigns them to redo workers for processing
`db2redow`	Processes redo log records during recovery at the request of the redo master; multiple redo workers can exist
`db2logmgr`	Log manager. Manages log files (archives/retrieves) for a recoverable database. Also does log file pre-allocation for recoverable databases
`db2logalloc`	Asynchronous log file allocator (added in DB2 10.5)

Just for interest, these are the different EDUs (engine dispatchable units) – i.e.

threads – that are related to logging. If you look at the EDUs running in a DB2 environment using something like db2pd -edu then you're sure to see some of these things.

Database Configuration Parameters for Logging

- NEWLOGPATH (Log Path)
- MIRRORLOGPATH
- OVERFLOWLOGPATH *

- LOGRETAIN *(removed in DB2 10.1)*
- USEREXIT *(removed in DB2 10.1)*
- LOGARCHMETH1/2 *
- LOGARCHCOMPR1/2 * *(new in DB2 10.1)*
- LOGARCHOPT1/2 *
- NUMARCHRETRY *
- ARCHRETRYDELAY *
- FAILARCHPATH *

- CUR_COMMIT

- LOGBUFSZ **
- LOGPRIMARY **
- LOGSECOND * **
- LOGFILSIZ **

- BLK_LOG_DSK_FUL *
- MAX_LOG *
- NUM_LOG_SPAN *
- MINCOMMIT * ** *(ignored in DB2 10.1)*
- BLOCKNONLOGGED *

- LOG_APPL_INFO *(new in DB2 10.1)*
- LOG_DDL_STMTS * *(new in DB2 10.1)*

- SOFTMAX ** *(deprecated in DB2 10.5)*
- PAGE_AGE_TRGT_MCR * *(new in DB2 10.5)*
- PAGE_AGE_TRGT_GCR * *(new in DB2 10.5)*

* Can be changed dynamically
** Updated by Configuration Advisor (AUTOCONFIGURE)

To give you an idea of how important logging is to DB2, these are all of the

logging-related database configuration parameters (and none of the HADR-related ones are shown either – but HADR is related to logging and recovery as well). Ones with blue asterisks can be changed dynamically and ones with a double red asterisk get updated by default when you create a new database. Some parameters come and go and this has been reflected on the slide.

LOGARCHCOMPR1/2 can be set to ON or OFF (OFF is default)

The parameters marked as dynamic (configurable online) and updated by configuration advisor are based on the state of things in DB2 11.1.1.1. This might change in the future.

What is Rollforward Recovery?

- **Rollforward recovery brings the database to a consistent state following a restore**
  - Transaction log records are used to redo work and then subsequently undo any in-flight changes that exist at the end of the rollforward
- **Only supported for recoverable databases**
- **Database rollforward**
  - Can be performed following a database restore
  - Can rollforward to end-of-logs or to a point-in-time
  - Offline operation
- **Table space rollforward**
  - Can be performed following a table space restore
  - Can rollforward to end-of-logs or to a point-in-time (at least to MRT)
  - Can be online

<No speaker notes for this slide>

What is Crash Recovery?

- **Crash recovery brings the database to a consistent state following a "crash" (abnormal termination) of the database**
  - Also known as "restart recovery"
- **Crash recovery is performed:**
  - Explicitly by the user using `RESTART DATABASE` command – or –
  - Implicitly by the database during first connect if `AUTORESTART= ON`
- **Occurs in two phases**
  - Redo phase: Transaction logs are used to redo work that may not have yet been persisted to disk
  - Undo phase: Uncommitted (in-flight) work is rolled back
- **Crash recovery is an offline operation**
  - Connections are blocked until recovery completes

<No speaker notes for this slide>

What is Crash Recovery? (cont.)

- **In DPF, crash recovery done on a per-database partition basis**
  - If non-catalog partition being recovered, will subsequently connect to catalog partition, driving crash recovery there if necessary
- **In pureScale, two types of crash recovery exist**
  - **Member crash recovery:** Individual recovery of member after member failure
  - **Group crash recovery:** Recovery of entire cluster after cluster failure

<No speaker notes for this slide>

Parallel Recovery

- **Log records are replayed in parallel by multiple recovery agents**
- **Applies to both crash recovery and rollforward recovery**
- **# of recovery agents is equal to # of logical CPUs**
  - Minimum of 3 recovery agents
  - One becomes the redo master, remainder are redo workers
  - Log records are read and placed onto queues for workers to handle

16

At runtime, work is often being done in parallel by multiple users/applications. To

ensure that recovery is fast, it is parallelized as well.

Monitoring Recovery Progress

- **LIST UTILITIES SHOW DETAIL**

```
ID                                = 1
Type                              = CRASH RECOVERY (or ROLLFORWARD RECOVERY)
Database Name                     = SAMPLE
Member Number                     = 0
Description                       = Crash Recovery (or Rollforward Recovery)
Start Time                        = 04/29/2018 15:20:05.646020
State                             = Executing
Invocation Type                   = User
Progress Monitoring:
   Estimated Percentage Complete = 75
   Phase Number [Current]         = 1
      Description                 = Forward
      Total Work                  = 163834 bytes
      Completed Work              = 124145 bytes
      Start Time                  = 04/29/2018 15:20:05.646121
   Phase Number                   = 2
      Description                 = Backward
      Total Work                  = 163834 bytes
      Completed Work              = 0 bytes
      Start Time                  = Not Started
```

- **Similar information also available through db2pd `-recovery`**

This is example output for the LIST UTILITIES SHOW DETAIL command.

Log Paths and Metadata Files

- **Default log path**
  - 9.7: `<dbPath>/<instance>/NODE####/SQL#####/SQLOGDIR`
  - 10.1+: `<dbPath>/<instance>/NODE####/SQL#####/LOGSTREAM####`
- **Non-default log path (if changed using `NEWLOGPATH`)**
  - 9.7: `<newLogPath>[/NODE####]`
  - 10.1+: `<newLogPath>/NODE####/LOGSTREAM####`
  - The log path contains a tag file called `SQLLPATH.TAG`
- **Database directory contains metadata files associated with logging**

```
<dbPath>/<instance>/NODE####/SQL####/SQLOGCTL.GLFH.1/2
<dbPath>/<instance>/NODE####/SQL####/MEMBER####/SQLOGCTL.LFH.1/2
                                              .../SQLOGMIR.LFH
```


KS-INTERNAL: Validated for 11.1.1.1.

For reference, these are the locations of the default log path and the directory structure that gets created when you specify a new log path.

Log Paths

- **Current active log path**
  - Read-only configuration parameter that shows path to active log files
  - Default location is within the database directory
  - Changed by setting `NEWLOGPATH` database configuration parameter
  - Raw logs are deprecated (so avoid using)
    - Raw logs can't be used with pureScale
- **Mirrored log path**
  - Second set of log files are maintained in a separate directory from the main log path
  - Not done by default – enabled by using the `MIRRORLOGPATH` database configuration parameter
  - Minor overhead associated with maintaining two sets of log files

Some internal tests of the impact of log mirroring showed that with a “good” placement (separate disk spindles) that there was a TPS drop of ~5-7%. With a very poor placement (on the same spindles as the primary log) there was a drop of ~12-15%. This was with a medium-heavy OLTP test workload, 7000 tps, 80% read & 20% write, 16-way Power6, AIX 6, DB2 V9.7 FP2, DS4700 SAN controller, 96 disks.

Log Paths (cont.)

- **Archive log path(s)**
  - Can configure one or two target locations for saving log files
  - Set using `LOGARCHMETH1` and `LOGARCHMETH2` database configuration parameters
  - Supports disk path, TSM, or vendor library
  - Logs are eligible for archiving when full or after truncation (e.g. online backup, `ARCHIVE LOG` command, clean database shutdown)

- **Overflow log path**
  - Used to specify where archived log files can be found (beyond typical archive location)
  - Can be set for the database using the `OVERFLOWLOGPATH` database configuration parameter
  - `ROLLFORWARD` and `RECOVER` commands also have an `OVERFLOW LOG PATH` option
  - Often used to manually "prefetch" retrieval of log files to speed up rollforward recovery

<No speaker notes for this slide>

db2cklog – Validate Archive Log Files

- **Check archived log files to ensure rollforward recovery will be successful**
- **Can check a single log file or range of log files**
- **Keystore arguments are required if the log files are encrypted**

```
db2cklog (DB2 Check Log File tool)
------------------------------------------------------------------------------
Syntax: DB2CKLOG [ CHECK ] <log-file-number1> [ TO <log-file-number2> ]
        [ ARCHLOGPATH <archive-log-path> ]
        [ KSPASSWORD <keystore-password> | KSPASSARG fd:<file-descriptor> |
          KSPASSARG filename:<filepath>  | KSPROMPT ]
```

```
> db2cklog check 4

=======================================================
"db2cklog": Processing log file header of "S0000004.LOG".

"db2cklog": Processing log pages of "S0000004.LOG" (total log pages: “144").
            ==> page "1" ...
            ==> page "101" ...

"db2cklog": Finished processing log file "S0000004.LOG". Return code: "0".
=======================================================
```


KSPASSWORD *keystore-password* Specifies the password to use when opening the keystore.

KSPASSARG fd:*file_descriptor* | filename:*filepath* Specifies the keystore password arguments. The *file_descriptor* parameter specifies a file descriptor that identifies an open and readable file or pipe that contains the password to use. The *filepath* parameter specifies the path of the file that contains the password to use.

KSPROMPT Specifies that the user is to be prompted for a password.

Agenda

- Logging Basics and Concepts
- Log Consumption
- Configuring the Active Log Space
- Monitoring Logging Activity

The presentation is broken up into three main sections that will cover the following

topics:

- Understand the basics and concepts associated with logging and recovery in DB2 LUW
- Determine how to configuring logging (for feature and performance)
- Describe the active log space and how to configure it
- Determine how to limit what gets logged and how to minimize crash recovery time
- Learn how to monitor logging activity for the database and individual transactions

Log Consumption – INSERT and DELETE

- **Row images are logged so that DB2 can redo or undo actions**
  - Real log space from active log is written and consumed
  - Virtual log space from active log is reserved for rollback
- **INSERT**
  - Row image being inserted is logged as part of a single “insert row” log record (required for redo!)
    - Non-inlined LOBs, long field columns that are stored outside of the base data object are logged separately
  - Reserve log space for “delete” on undo
    - Space for row image is not required in reserved space
- **DELETE**
  - Row image being deleted is logged (required for undo!)
  - Reserve log space for “insert” on undo
    - Space for row image is required in reserved space
- **When row compression is active, the row images are compressed, resulting in fewer bytes logged, reserved, and less log files usage**

23

<Slide courtesy of Mike Winer>

This is for row-organized tables.

Log Consumption – UPDATE

- **There are three different types of UPDATE log records written by DB2:**

  1. **Full before and after row image logging.** The entire before and after image of the row is logged. This is the only type of logging performed on tables enabled with DATA CAPTURE CHANGES.

  2. **Full XOR logging.** The XOR differences between the before and after row images, from the first byte that is changing until the end of the smaller row, then any residual bytes in the longer row.

  3. **Partial XOR logging.** The XOR differences between the before and after row images, from the first byte that is changing until the last byte that is changing. Byte positions may be first/last bytes of a column. Row images must be the exact same length.

<Slide courtesy of Mike Winer>

This is for row-organized tables.

Log Consumption – UPDATE Examples

1. Full before and after row image logging (DATA CAPTURE CHANGES)

B bytes

Fred	500	10000	Plano	TX	24355

→

A bytes

John	500	10000	Plano	TX	24355

Logged: 20 + 20 + B + 20 + A

Fred	500	10000	Plano	TX	24355

John	500	10000	Plano	TX	24355

Reserved: 32 + 20 + B [+ 34 if B>A]

2. Full XOR logging (row length changing updated)

B bytes

Fred	500	10000	Plano	TX	24355

→

A bytes

Frank	500	10000	Plano	TX	24355

Logged: 20 + 22 + MAX(A,B) – 2 (offset of first byte changed)

1101101010010100110010010100101010101101001010101101010101010	01

Reserved: 26 + 22 + B – 2(offset of first byte changed) [+ 34 if B>A]

3. Partial XOR logging (row length does not change)

B bytes

Fred	500	10000	Plano	TX	24355

→

A bytes

John	500	10000	Plano	TX	24355

Logged: 20 + 24 + (4)

110110101001010011

Reserved: 26 + 24 + (4)

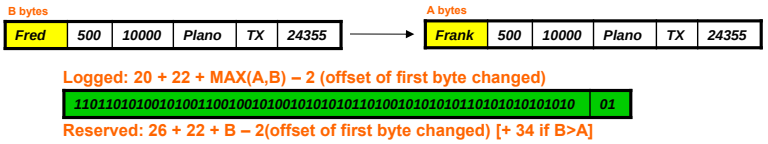
<Slide courtesy of Mike Winer>

This is for row-organized tables.

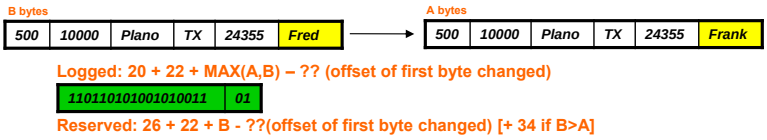
Log Consumption – Full XOR Logging Details

- When the total length of the row is changing, which is common when variable length columns are updated and also when row compression is enabled, DB2 will determine which byte is first to be changing and log a Full XOR log record.

1. Full XOR logging (length change) with changed column at/near beginning of row



2. Full XOR logging (length change) with changed column at/near end of row



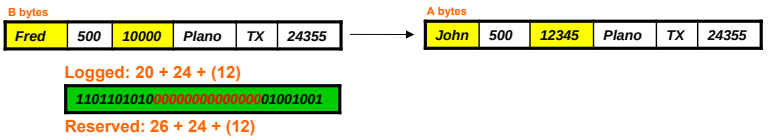
<Slide courtesy of Mike Winer>

This is for row-organized tables.

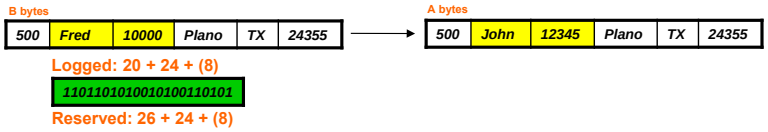
Log Consumption – Partial XOR Logging Details

- When the total length of the row is not changing, even when row compression is enabled, DB2 will compute and write the most optimal Partial XOR log record possible.

1. Partial XOR logging (no length change) with a gap between columns being changed



2. Partial XOR logging (no length change) with no gap between columns being changed



<Slide courtesy of Mike Winer>

This is for row-organized tables.

Log Consumption – Column Ordering Summary
(for Row-organized Tables)

- **Columns which are updated frequently (changing value) should be:**
  - Grouped together
  - Defined towards or at the end of the table definition
- **These recommendations are independent of**
  - Row compression
  - Row format (default or null/value compression)
- **The benefit would be:**
  - Better performance
  - Less bytes logged
  - Less log pages written
  - Smaller active log requirement for transactions performing a large number of updates

28

<Slide courtesy of Mike Winer>

This is for row-organized tables.

Log Consumption – Column-Organized Tables

- **INSERT**
  - Inserted value for each column is logged separately in its own log record
    - As each column has its own set of pages and each log record reflects an insert into a page
  - If multiple inserts are performed within the same transaction then all of the row values for column 1 are logged together in a single log record, all values for column 2 are logged together, etc.
    - More specifically, all values for a given column that end up on the same page are logged together
- **DELETE**
  - Logical delete (and logging of logical delete) where the row is marked as deleted, but column values not physically deleted from pages
- **UPDATE**
  - Effectively handled as a DELETE and INSERT

Agenda

- Logging Basics and Concepts
- Log Consumption
- Configuring the Active Log Space
- Monitoring Logging Activity

<No speaker notes for this slide>

Configuring the Active Log Space

- **Active log space is determined by**
  - `((# primary log files + # secondary log files) x log file size)`
- **At a minimum, total active log space should be configured to accommodate**
  - All log records generated by the longest running transaction
  - Log records generated by all transactions executed concurrently within the span of the longest running transaction
- **Allocate more to avoid "out of log space" errors during peak usage**
  - Strategies can be employed to limit impact of long running transactions and affect on crash recovery time (e.g. `NUM_LOG_SPAN, MAX_LOG`)
- **With a total size chosen, pick an appropriate log file size and then determine the number of log files needed**

31

The maximum size for the active log space in DB2 10.5 (and earlier) is 1 TB (256

log files x 4 GB (maximum log file size in 10.5 and earlier)).

The maximum size for the active log space in DB2 11.1 is 16 TB (256 log files x 16 GB (maximum log file size in 11.1)).

Choosing the Log File Size

- **Size of each primary & secondary log file determined by `LOGFILSIZ` database configuration parameter**
- **Maximum log file size:**
  - V9.7-V10.5: 4 GB
  - V11.1: 16 GB
- **Consider the following when choosing the size:**
  - Frequency of archiving required (e.g. for DR and log shipping)
  - Limit of 256 logs so very large active log space requires large files
  - Very small size means very frequent log file allocation which may cause performance blips and lower overall scalability
  - The larger the file, the longer it takes to physically allocate it, so applications may have to wait longer before a file can be used
    - DB2 tries to avoid this where possible by pre-allocating and renaming old files instead of allocating
- **Minimum suggested size is 20 - 50 MB**
- **Large is good provided that archiving requirements, etc. are met**

<No speaker notes for this slide>

Number of Log Files

- **Determined by `LOGPRIMARY` and `LOGSECOND` database configuration parameters**
- **`LOGPRIMARY`: Number of primary log files**
  - Pre-allocated log files, always exist
- **`LOGSECOND`: Number of secondary log files**
  - Allocated on-demand when not enough space available in primary logs (and freed over time as they are no longer required)
  - Recommended setting is `0` (allocate all log files as primary)
  - Can be set to `-1` to enable infinite logging (more on this later)
- **Rule: `(LOGPRIMARY + LOGSECOND) <= 256`**

The rule obviously assumes a setting other than -1 for LOGSECOND. If

LOGSECOND is -1 then LOGPRIMARY must be <= 256.

Configuring the Log Buffer Size

- **Specifies size of memory buffer used to hold log records before they are written to disk**
  - Set using `LOGBUFSZ` database configuration parameter
- **Log records are written to disk when one of the following actions occur**
  - A transaction commits
  - The log buffer is full
  - A data page is written disk (requiring all log records up to the page LSN to be written out to disk as well)
  - Some other internal database manager event
- **Size of buffer impacts performance for logging, rollback, and currently committed processing**
  - Impact is more significant for OLTP workloads
- **Also want buffer to be large enough that most rollbacks can be serviced from the buffer – this avoids hitting the log disk for reads which can impact write performance**

<No speaker notes for this slide>

Configuring the Log Buffer Size (cont.)

- **Optimal value depends on workload and amount of concurrent Insert/Update/Delete activity**
- **Default (1MB) is way too low, but can be auto-configured**
- **Consider setting to at least a couple of MB, but something in the 16MB – 64MB range is reasonable to start with**
- **Monitor for "log buffer full" conditions**
  - Watch `NUM_LOG_BUFFER_FULL` monitor element, want to keep this value low
- **Bigger isn't always better**
  - A very large buffer could hurt performance in some cases, so test!
- **Allocated from database heap, so adjust database heap accordingly**

<No speaker notes for this slide>

Configuring the Log Path Storage

- **Avoid sharing storage with other database objects**
  - Use dedicated storage with sufficient throughput capacity
- **RAID-5 or RAID-10 (preferred) for best protection and performance**
  - Small strip size – e.g. 8KB
  - Spindles matter – don't skimp on the drives
- **Enable storage write cache and cache mirroring on the SAN**
- **SSDs for logs may or may not help the performance of the system**
  - Log I/Os may hit the storage cache anyway
  - However, if system is log bound then it could make a big difference
    - Long commit times, long log write times
  - Not usually the case for warehouse environments, but more common in OLTP
- **Raw logs have been deprecated – so best not to use them**
  - Direct I/O has similar performance characteristics
    - Used for runtime processing by default starting in 9.7
    - See `DB2_LOGGER_NON_BUFFERED_IO` registry variable for details
      - Default is `AUTOMATIC`, which means active logs are open as non-buffered (direct) but other access (crash recovery, rollforward, archiving) uses buffered I/O – Default is typically best

(KS-INTERNAL section below after this public information)

We tend to lean more to RAID-10 over RAID-5; while writes to either array will result in at least 2 drives being hit (1+p vs 1+1), RAID-10 has performance advantages when there are rollbacks that need log records to be read from disk, as the reads can be serviced by the mirror copies and won't impact write performance as much as it would on RAID-5.

In V9.7, Enhanced DB2_LOGGER_NON_BUFFERED_IO capabilities:

- OFF: Files always opened for buffered I/O
- ON: Files always opened for direct (non-buffered) I/O
- AUTOMATIC (new default): DB2 determines what log files would benefit from non-buffered I/O based on how they are being used:
  - Runtime has active logs non-buffered
  - Archiving opens them buffered
  - Crash/rollforward recovery opens them buffered

KS-INTERNAL:

With respect to the statement about "if log writes are bottlenecking the system", this means that the system is suffering from long commit times and increased log I/O latency (long write times).

CONFIDENTIAL: Steve: In-house testing at one point in the past showed that one drive could cope with about 400 transactions per second of a TPC-C-like workload (but it is workload dependant). Sunil: During benchmarking we have driven greater than 250 MB/second of logging. Peak customer usage we have seen is about 60 MB/second in a stock exchange environment. Typically it is in the low 10's of MB/second. Peter S: When it comes to how many log disks are needed, the answer is whatever number sufficient to achieve the < 1ms access time, to support the MB/sec that will be written to log, based on the number of transactions per second and the average size of the log records.

Historically we have recommended RAID-10 but we tend to recommend RAID-5 now. Spindles matter and typically in SANs writes are going into the disk cache anyway. In our own "extreme tuning" for TPC-C we may end up using RAID-5 or RAID-10 based on absolutely best performance, which depends on the storage controller we use.

Cache mirroring is used for durability. There is overhead of cache mirroring, but there's no short cut here or ways of getting around it.

From Sunil Kamath's article "Unleashing the Value of Solid-State Drives for DB2 Workloads":

> *"Finally, active log files are less desirable candidates for placement on SSDs, as these operations usually drive sequential I/Os and are typically hit on storage write cache. However, if the application is suffering from large commit times and increased log I/O latency, placing the active logs on SSDs can help."*

Buffered I/O:

- Prior to V9.7, buffered I/O is the default. Write-thru is enabled, however, to ensure log records make it to storage, but writes

- Pre-V9.5 FP1: Log file system could be mounted with DIO/CIO options
- V9.5 FP1+: DB2_LOGGER_NON_BUFFERED_IO registry variable, so no longer need to use file system mount options

  OFF (default): Files always opened for buffered I/O

  ON: Files always opened for direct (non-buffered) I/O

Non-buffered logs may result in degraded performance for:

- Archiving logs (although this is asynchronous and archive target might be the bottleneck anyway – e.g. TSM)
- Rollback (can be addressed by tuning the size of the in-memory log buffer – LOGBUFSZ)

Information here provided from Steve Rees and Matt Emmerton.

Infinite Logging

- **No limit on the amount of log space that can be consumed by active transactions**
- **Enabled by setting `LOGSECOND` to `-1` (can be set dynamically)**
- **Database must also be configured for archive logging**
  - By setting `LOGARCHMETH1` to one of `DISK`, `TSM`, `VENDOR` or `USEREXIT`
- **A log file is archived once it has been filled, but may be kept in active path longer (to avoid possible retrievals)**
- **Consider setting `MAX_LOG` and `NUM_LOG_SPAN` to prevent errant applications from keeping the first active log file too far in the past**
  - `MAX_LOG` – Maximum percentage of primary log space one transaction can use
  - `NUM_LOG_SPAN` – Number of active log files that a transaction can span
- **Incompatible with HADR environment**

MAX_LOG: Limits the total amount of primary log space that a transaction can consume, including space consumed by log records already written and space reserved in case of transaction rollback. It is specified as a percentage: 0-100 (0 means no limit; default). If reached, the application is forced off, receiving an SQL1224N error and the transaction is rolled back. The DB2_FORCE_APP_ON_MAX_LOG registry variable can be set to FALSE in which case an application gets an SQL0964N ("transaction log is full") error instead. The application can commit work done by previous successful statements. Utilities such as reorg, load, and backup ignore this parameter

NUM_LOG_SPAN: Limits the number of active log files that a transaction can span. Only considers actual log records written (and not space reserved for transaction rollback). The range is 0 – 65535 (0 means no limit; default). If reached, the application is forced off, receiving an SQL1224N error and the transaction is rolled back. Utilities such as reorg, load, and backup ignore this parameter

Infinite Logging: Implications of Use

- **Log files may need to be retrieved during rollback and crash recovery**
  - Depending on number of log files to process (and need for retrieval) it may take a very long time to perform crash recovery
- **Compensation log space is not reserved for potential rollback of transactions**
  - A full log path and archive location can result in failures trying to write undo log records, bringing the database down
  - `BLK_LOG_DSK_FUL` should be enabled to avoid this

38

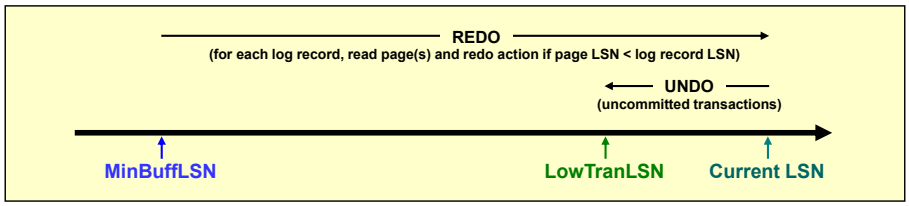
This is very important to note if you're using infinite logging. There have been cases where people have enabled it for some large batch workload and forgot to disable it. A bug in an application that opens a transaction but doesn't commit it can lead to the log space growing and growing – which is a problem if the database ever has to go through crash recovery after this. Depending on number of log files to process (and

need for retrieval) it may take a very long time to perform crash recovery.

Also, one thing that could be impacted but isn't listed here is the currently committed semantics. With currently committed, a log record is read from buffer or the log file on disk to find the last committed value of a row that is being updated (avoiding the need to lock the row). If the log file has been archived then it will not be retrieved and the application will wait on the row lock instead.

Tuning Crash Recovery (Pre-DB2 10.5)

- **Redo starting point is minimum of (MinBuffLSN, LowTranLSN)**
  - MinBuffLSN represents the oldest change to a page in the buffer pool that has not been written to disk yet
  - LowTranLSN represents the oldest active uncommitted transaction (specifically, the LSN of the first log record it wrote)



- **SOFTMAX configuration parameter influences how far back MinBuffLSN is**
  - Value is percentage of one log file (for example, 200 = 2 log files)
  - Page cleaners triggered to write pages out if MinBuffLSN too far back ("LSN gap")
  - Lower value means shorter recovery time, but increased page cleaning activity
  - Higher value means longer recovery time, but fewer page writes
  - SOFTMAX deprecated in DB2 10.5 (more details later)

39 © 2019 IBM Corporation

For non-pureScale environments, if the system fails then a crash recovery is necessary to bring the database to a consistent state (ensuring committed transactions have been persisted to the database). This is done through the process of crash recovery (also known as restart recovery).

DB2 will start the recovery at a particular point in the past within the log files. Specifically, it will start at the minimum of the MinBuffLSN and LowTranLSN values. These values are maintained at runtime and are periodically flushed to disk so that they can be used if/when crash recovery happens.

MinBuffLSN represents the oldest change to a page in the buffer pool that has not been written to disk yet (typically this is associated with a transaction that has already been committed since then, but does not necessarily have to be; it's simply a matter of age)

LowTranLSN represents the oldest uncommitted transaction that is currently active in the system.

Crash recovery involves doing two passes or phases. The first is redo, which replays the log records to ensure that committed transactions are persisted in the database. As log records are replayed during redo, the data page(s) that corresponds to the change is read in and the page LSN is looked at. If the page LSN is less than the LSN of the log record then the work is redone, otherwise it isn't (because we know that the page already reflects the change). Once the redo phase is completed then the undo phase is performed. In this case, all of the transactions that are found to be uncommitted are undone (rolled back).

In an OLTP system, transactions are typically short and so MinBuffLSN is usually further in the past than LowTranLSN is. However, if LowTranLSN was further back (because of a really long transaction that was still active) then the recovery would start there, but it wouldn't have to do any work in log records prior to MinBuffLSN because we know their changes have been persisted to disk already. We just have to start back there because we need to know all about the uncommitted transactions that were running and that involves starting at the oldest of them.

Now, assuming that all of the applications are well behaved (short transactions) then it is likely that MinBuffLSN will be further back than LowTranLSN and therefore it will dictate the starting point for recovery. How far back MinBuffLSN falls is configurable using the SOFTMAX database configuration parameter. The value corresponds to the percentage of one log file. For example, if you set it to 200 then that means that DB2 will typically try to keep MinBuffLSN about two log files back from the Current LSN being written – which means that we would have to redo two log files' worth of log records during crash recovery. When MinBuffLSN falls outside of this range (called an LSN gap) then the page cleaners are triggered (LSN gap trigger) to clean those older pages.

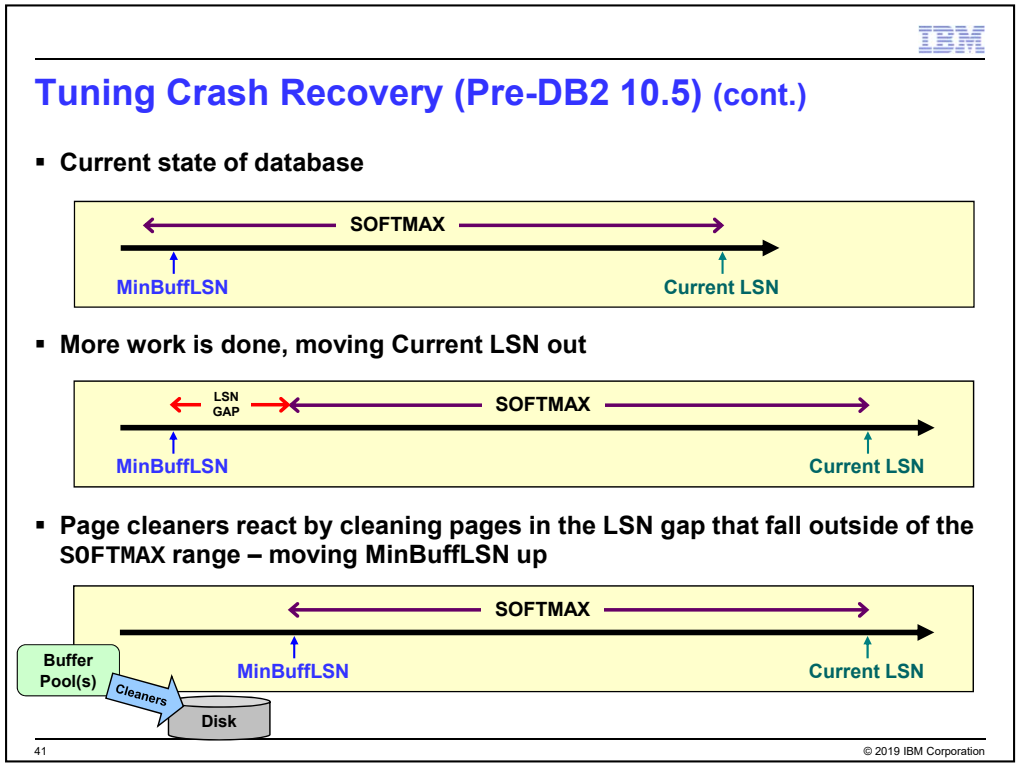
Note that setting SOFTMAX has both an impact on crash recovery time but also potentially on runtime performance. A lower value means shorter recovery time, but increased page cleaning activity. On the other hand, a higher value means longer recovery time, but fewer page writes. The latter is likely to mean better runtime performance. Why? With a very low value and the resulting increase in page cleaning, consider an example where a page is frequently updated. It could be that after every couple of updates that the page gets written out to disk, only to be updated and dirtied a short time later. From a runtime perspective, the last page flush effectively could have persisted all of the changes made to the page before now making all previous writes unnecessary (with an unnecessary load on the disks). Of course, the flip side to that is that if those previous writes had not occurred

and the database was terminated abnormally, then crash recovery could take longer to run.

Tuning Crash Recovery (Pre-DB2 10.5) (cont.)

- **LowTranLSN is in your hands**
  - Based on age of the transactions
  - Keep transactions short, commit frequently
- **If LowTranLSN < MinBuffLSN then DB2 will still replay those log records in between, but only for the purposes of rebuilding the transaction table**
  - All data changes have already been persisted to disk, so no need to replay them

<No speaker notes for this slide>



This is an example illustrating how SOFTMAX impacts where MinBuffLSN falls within the log stream.

Step #1. MinBuffLSN falls within the SOFTMAX range/window (again, the SOFTMAX range is some number of log files back from the point we are currently writing to in the logs).

Step #2. Work is done against the database which is resulting in log records being written and us moving further along in the logs. With Current LSN moving further out, the SOFTMAX window moves as well, leaving MinBuffLSN outside of it. Again, that difference between MinBuffLSN and the edge of the SOFTMAX range is called an LSN gap.

Step #3. Because we don't want to have to do more work during crash recovery than what is within the SOFTMAX range, we need to move MinBuffLSN up. This is done by posting an LSN gap trigger to the page

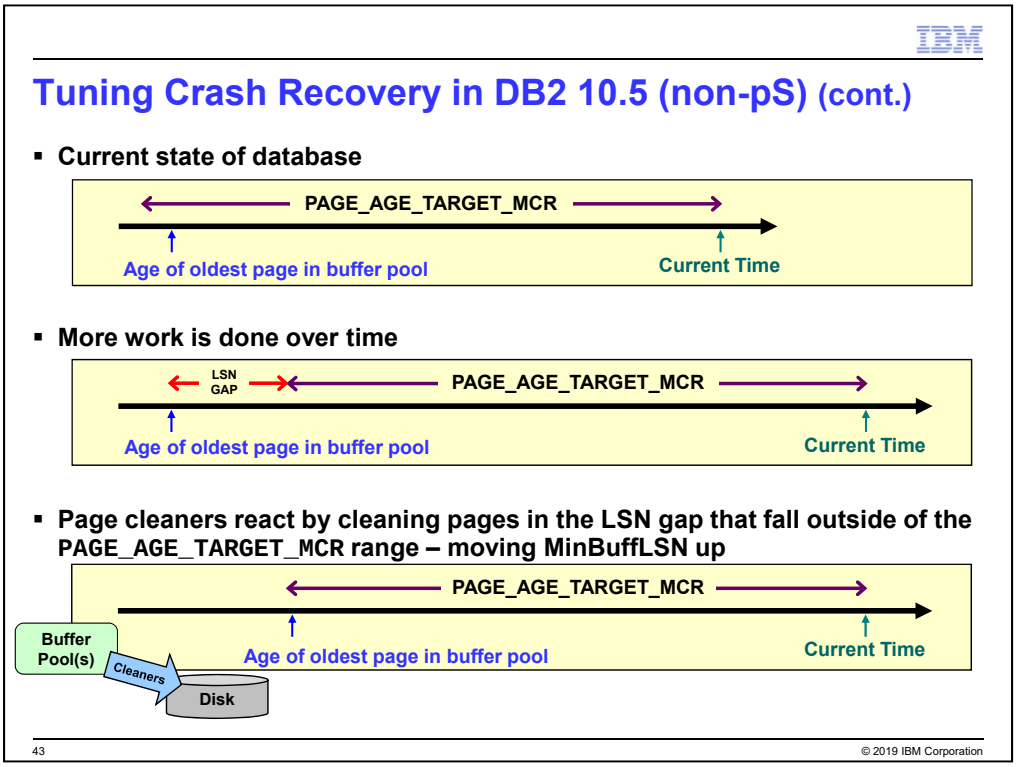
cleaners. They will clean the older pages from out of the buffer pool to disk to the point where we can move MinBuffLSN back into the SOFTMAX range.

Note that this is using the default page cleaning algorithm in DB2. An alternate page cleaning algorithm also exists (sometimes referred to as proactive page cleaning), which is enabled via the DB2_USE_ALTERNATE_PAGE_CLEANING registry variable. The proactive page cleaning method modulates the default behavior by distributing the same number of writes over a longer period of time. The page cleaners do this by cleaning not only the pages that are contributing to an LSN gap, but also pages that are likely to contribute to an impending LSN gap, based on the current level of activity.

Tuning Crash Recovery in DB2 10.5

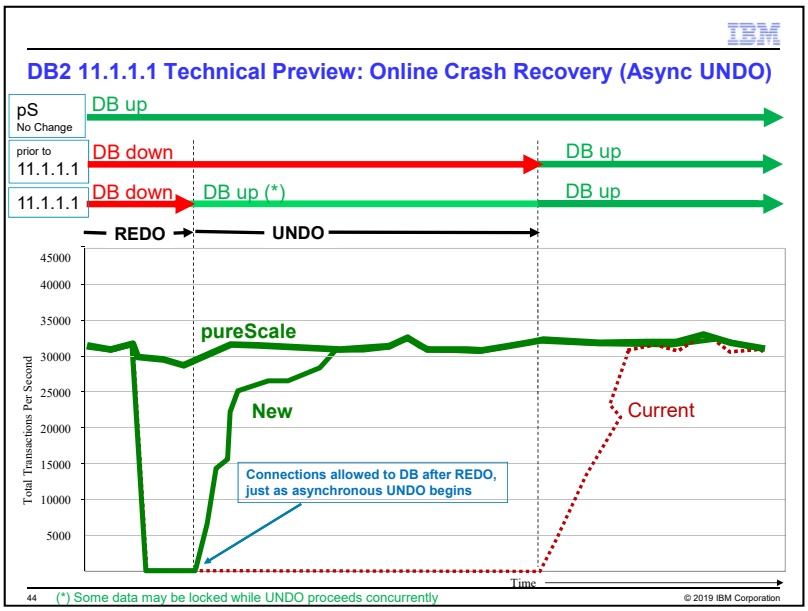
- **Same principles apply, but new database configuration parameters replace `SOFTMAX`**
  - `SOFTMAX` deprecated as of DB2 10.5
  - `SOFTMAX=0` means use new parameters
    - Set by default for new databases created in DB2 10.5
    - Old value maintained for upgraded databases
- **`PAGE_AGE_TRGT_MCR`**
  - Target duration (in seconds) for changed pages to be kept in the local buffer pool before being persisted to disk (or to the group buffer pool (GBP) in pureScale)
  - Applicable to both non-pureScale and pureScale environments
- **`PAGE_AGE_TRGT_GCR`**
  - Target duration (in seconds) for changed pages to be kept in the GBP before being persisted (cast out) to disk
  - Applicable to pureScale environments only

SOFTMAX has been deprecated in DB2 10.5. New PAGE_AGE_TRGT_MCR and PAGE_AGE_TRGT_GCR database configuration parameters have been added instead.



This is similar to the example shown earlier but illustrating here how PAGE_AGE_TARGET_MCR impacts how old the pages are that are kept around in the buffer pool(s).

No example is shown for pureScale, but Member Crash Recovery behaves as shown here. Group Crash Recovery is similar – except that it involves the group buffer pool instead of local buffer pools, a merged log stream (with all of the logs across the members) instead of a single member's log stream, and the PAGE_AGE_TARGET_GCR configuration parameter.



This graph shows the availability characteristics of a DB2 database (with pureScale, non-pureScale prior to 11.1.1.1, and now non-pureScale in 11.1.1.1) following a crash – when DB2 is performing crash recovery.

We'll focus on the non-pureScale example here, as pureScale had already been designed to offer maximum database and data availability in case of a crash.

DB2 has a REDO and an UNDO phase of crash recovery. REDO ensures that all activities that had occurred prior to the crash (for log records that were written to the log files – for example, committed transaction activity) are replayed and in the database as if the crash had not occurred. The UNDO pass then rolls back all uncommitted transactions that were in flight at the time of the crash. Connections to the database are not allowed until both the REDO and UNDO phases of crash recovery are completed.

Now in DB2 11.1.1.1, DB2 will **allow connections to the database after the REDO phase and during/while the UNDO phase** executes. Data (tables) that have uncommitted activity to be undone will be locked, but access to all other tables will be allowed. This is of most benefit in batch/ETL processing environments where a large batch job which inserted/updated/deleted many many rows will take some time to undo – this should not prevent applications from connecting to the database while the large and uncommitted transaction is being rolled back in the crash recovery UNDO phase.

See this technote for details: http://www-01.ibm.com/support/docview.wss?uid=swg21994342.

Agenda

- Logging Basics and Concepts
- Log Consumption
- Configuring the Active Log Space
- Monitoring Logging Activity

<No speaker notes for this slide>

Monitoring Logging Activity

- **Many different things that can be monitored**
  - Log full situations
  - Overall database log usage
  - Log space usage at the transaction level
  - Log reads and writes
  - Time spent metrics for logging
- **Sources of information include:**
  - Monitor functions
  - Monitor views
  - db2pd
  - db2diag.log and notification log

<No speaker notes for this slide>

Monitor Views/Functions

- **`MON_GET_TRANSACTION_LOG` table function**
  - **Added in DB2 10.1**
  - Returns information about the transaction logging subsystem
  - Includes important elements from deprecated `SYSIBMADM.SNAPDB` and `SYSIBMADM.SNAPDETAILLOG` admin views
  - Also includes information previously only available through `db2pd`
    - Currently committed metrics
    - Archive logging status
    - Current LSN
    - Current log chain
    - LSN of oldest active transaction
- **`MON_TRANSACTION_LOG_UTILIZATION` admin view**
  - **Added in DB2 10.5**
  - Returns information about transaction log utilization
  - Includes same elements from deprecated `SYSIBMADM.LOG_UTILIZATION` admin view such as total log space used and available

DB2 10.1 and 10.5 introduced new administrative views related to monitoring.

Log Full Situations

- **When log full (SQL0964N) situations arise, information can be found in the notification log and db2diag.log**
- **Older dirty (modified) pages in the buffer pool(s)**
  - `ADM1822W  The active transaction log is being held by dirty pages. Database performance may be impacted.`
    `The rate at which the database work load is generating dirty pages has exceeded the rate at which the page-cleaner agents are writing dirty pages to disk. Dirty pages that have not yet been written to disk are holding the transaction log.`
  - `Corrective action:` Decrease the value of the PAGE_AGE_TRGT_MCR and PAGE_AGE_TRGT_GCR
- **Long running transaction(s)**
  - `ADM1823E  The active log is full and is held by application handle "###".  Terminate this application by COMMIT, ROLLBACK or FORCE APPLICATION.`

ADM1822W The active transaction log is being held by dirty pages.

Database performance may be impacted.

Explanation:

The rate at which the database work load is generating dirty pages has exceeded the rate at which the page-cleaner agents are writing dirty pages to disk. Dirty pages that have not yet been written to disk are holding the transaction log.

User response:

1. Reduce the database work load, if possible.
2. If this problem persists, take the following action:
   * Decrease the value of the PAGE_AGE_TRGT_MCR and PAGE_AGE_TRGT_GCR database configuration parameters
   * Increase the value of the NUM_IOCLEANERS database configuration parameter

Database Log Activity

- **Use the `MON_GET_TRANSACTION_LOG` table function**
  - One row per database partition/member is returned
- **`FIRST_ACTIVE_LOG`**
  - File number of the first active log file
- **`LAST_ACTIVE_LOG`**
  - File number of the last active log file
- **`CURRENT_ACTIVE_LOG`**
  - File number of the active log file that DB2 is currently writing to
- **first active <= current active <= last active**

MON_GET_TRANSACTION_LOG was added in DB2 10.1. The SYSIBMADM.SNAPDETAILLOG admin view can be used prior to DB2 10.1.

Database Log Space Usage

- **Use the `MON_TRANSACTION_LOG_UTILIZATION` monitor view**
  - One row per database partition is returned
- **`LOG_UTILIZATION_PERCENT`**
  - Percentage of the total active log space used
  - Calculated as `(TOTAL_LOG_USED/(TOTAL_LOG_USED + TOTAL_LOG_AVAILABLE))`
- **`TOTAL_LOG_USED_KB`**
  - Total amount of active log space currently used in the database
  - Includes space reserved for compensation (not applicable for infinite logging)
- **`TOTAL_LOG_AVAILABLE_KB`**
  - Amount of active log space in database currently available for use by transactions
  - Includes space from the primary log files and secondary log files (even if not allocated yet)
  - If newer transactions commit – but older uncommitted transactions still exist – then the log space used by those newer transactions is still considered used and is not available (but their compensation space is no longer reserved)
- **`TOTAL_LOG_USED_TOP_KB`**
  - Maximum amount of total log space that has been used in the database at one time

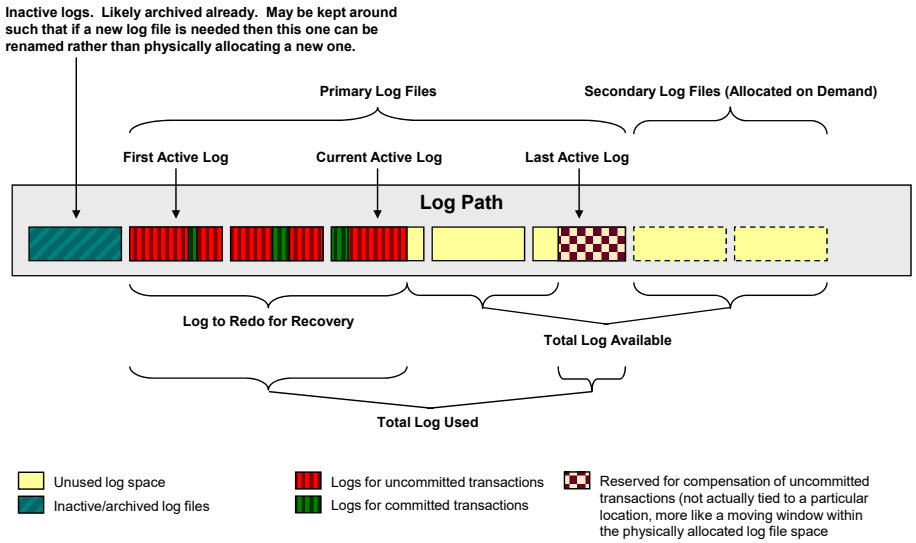
SYSIBMADM.MON_TRANSACTION_LOG_UTILIZATION was added in DB2 10.5. The SYSIBMADM.LOG_UTILIZATION admin view can be used prior to DB2 10.5.

Database Log Space Usage (cont.)

- **Use the `MON_GET_TRANSACTION_LOG` table function**
  - One row per database partition is returned
- **`TOTAL_LOG_AVAILABLE`**
  - Same as `TOTAL_LOG_AVAILABLE_KB` on previous page (but in bytes, not KB)
- **`TOTAL_LOG_USED`**
  - Same as `TOTAL_LOG_USED_KB` on previous page (but in bytes, not KB)
- **`LOG_TO_REDO_FOR_RECOVERY`**
  - Amount of log space (in bytes) that will have to be redone if the database is abnormally terminated and crash recovery is performed
  - Impacted by long running transactions and age of dirty pages in the buffer pool
    - See `APPL_ID_OLDEST_XACT` for oldest transaction running in the system
- **`LOG_HELD_BY_DIRTY_PAGES`**
  - Directly related to the age of the oldest dirty page in the database's buffer pool(s)
  - Amount of log space (in bytes) corresponding to the "distance" between the log record of the first update that dirtied that page and the current spot being written to in the logs
  - Impacted by the `PAGE_AGE_TRGT_MCR/PAGE_AGE_TRGT_GCR` database configuration parameters and page cleaning

MON_GET_TRANSACTION_LOG was added in DB2 10.1. The SYSIBMADM.SNAPDB admin view can be used prior to DB2 10.1.

Database Log Space Usage – Example



This diagram shows how database log space is logically organized.

Database Transaction Log Monitoring with db2pd

```
db2pd -db <dbname> -logs
```

```
Current Log Number              38
Pages Written                   257
Cur Commit Disk Log Reads       0
Cur Commit Total Log Reads      0
Method 1 Archive Status         n/a
Method 1 Next Log to Archive    n/a
Method 1 First Failure          n/a
Method 2 Archive Status         n/a
Method 2 Next Log to Archive    n/a
Method 2 First Failure          n/a
Log Chain ID                    0
Current LSO                     254577121
Current LSN                     0x0000000000272A17
```

Current active log file being written to

LSN where we'll start writing the next log record to; compare to a transaction's first LSN to see "distance" (in bytes) between start of transaction and the current logging location

```
Address            StartLSN           StartLSO           State      Size       Pages      Filename
0x00000000013D59A8 000000000026F1DB   253527201          0x00000000 1024       1024       S0000038.LOG
0x00000000013D6308 0000000000000000   257701025          0x00000000 1024       1024       S0000039.LOG
0x00000000013D6C68 0000000000000000   261874849          0x00000000 1024       1024       S0000040.LOG
0x00000000013D75C8 0000000000000000   266048673          0x00000000 1024       1024       S0000041.LOG
0x00000000013D7F28 0000000000000000   270222497          0x00000000 1024       1024       S0000042.LOG
```

Starting LSN can show what log file a particular transaction starts in

You can use db2pd to see information about logging at a database level.

Transaction Log Space Usage

- **Determine how much space has been used (already written, but not reserved) by all of the current write transactions**

```
SELECT CHAR(CONN.SYSTEM_AUTH_ID,10) AS AUTHID,
       CHAR(CONN.APPLICATION_NAME,10) AS APPNAME,
       INT(UOW.APPLICATION_HANDLE) AS APPHANDL,
       CHAR(UOW.WORKLOAD_OCCURRENCE_STATE,10),
       UOW.UOW_LOG_SPACE_USED AS LOG_SPACE,
       UOW.UOW_START_TIME
 FROM TABLE(MON_GET_UNIT_OF_WORK(NULL,NULL)) AS UOW,
      TABLE(MON_GET_CONNECTION(NULL,NULL)) AS CONN
 WHERE UOW.APPLICATION_HANDLE = CONN.APPLICATION_HANDLE AND
       UOW.UOW_LOG_SPACE_USED > 0
 ORDER BY UOW.UOW_LOG_SPACE_USED DESC

AUTHID     APPNAME    APPHANDL    4          LOG_SPACE   UOW_START_TIME
---------- ---------- ----------- ---------- ----------- --------------------------
KSCHLAMB   db2bp.exe          796 UOWEXEC          33354 2017-05-11-12.00.22.964269
KSCHLAMB   db2bp.exe          782 UOWWAIT           8460 2017-05-11-11.31.57.411029
```


Note that MON_GET_UNIT_OF_WORK is returning the amount of log space that has been consumed by log records written by the transaction so far – but it doesn't include the amount of log space reserved. To get that, use the db2pd -transactions command.

Transaction Log Space Usage (cont.)

```
db2pd -db <dbname> -transactions
```

LSN of the **first** log record written by the transaction; *(Smallest non-0 value indicates oldest write transaction)*

LSN of the **last** log record written by the transaction

AppHandl	TranHdl	State	Firstlsn	Lastlsn	LogSpace	SpaceReserved
782	14	WRITE	0x000000000027EB55	0x000000000027EBA2	8460	15591
780	15	READ	0x0000000000000000	0x0000000000000000	0	0
796	16	WRITE	0x000000000027EBB9	0x000000000027ECE9	33354	60498

Application handle can be used to get further information from monitor functions

Amount of log space used (but not reserved) by the transaction

Total log space used by the transaction including log records written and reserved space

<No speaker notes for this slide>

Monitoring Log Reads and Writes

- **Use the `MON_GET_TRANSACTION_LOG` table function**
  - One row per database partition/member is returned
- **Number of reads should typically be low – relative to writes**
- **`log_reads`**
  - # of log pages read from disk by the logger
  - Rollback, currently committed processing, etc.
- **`num_log_read_io`**
  - Number of read requests by the logger for reading log pages from disk
- **`log_read_time`**
  - Elapsed time spent by the logger reading log pages from disk
- **`num_log_data_found_in_buffer`**
  - Number of log page accesses that are satisfied by the log buffer
  - Configure `LOGBUFSZ` to improve log buffer hit ratio

MON_GET_TRANSACTION_LOG was added in DB2 10.1. The SYSIBMADM.SNAPDB admin view can be used prior to DB2 10.1.

KS-INTERNAL:

Pages read will not go into the shared in-memory log buffer that is used for writing log data out to disk. Instead, pages are read into internal memory buffers. The size

of the buffer used depends on internal algorithms and the number of concurrent read requests.

Monitoring Log Reads and Writes (cont.)

- **`log_writes`**
  - Number of log pages written to disk by the logger
  - Not the overall number of log pages produced
    - A particular page may be written multiple times as it is filled
- **`num_log_write_io`**
  - Number of write requests by the logger for writing log data to disk
- **`log_write_time`**
  - Elapsed time spent by the logger writing log data to the disk
  - Includes writing log data to both locations for mirrored logging

<No speaker notes for this slide>

Monitoring Log Reads and Writes (cont.)

- **`num_log_buffer_full`**
  - Number of times agents are unable to copy records into log buffer because it is full
    - Log buffer is flushed to disk and agents have to wait
  - Goal is to keep this value low
  - Possible reasons for it being high
    - `LOGBUFSZ` is too small for your environment
    - Workload consists of long running transactions with infrequent commits
    - Log storage cannot keep up with logging demands

- **`num_log_part_page_io`**
  - Number of write requests by the logger for writing a partial log page to disk
  - Included as part of `num_log_write_io` count as well
  - Usually indicates a lot of short frequently committing transactions
  - Informative value, not a lot of reason to monitor

<No speaker notes for this slide>

Monitoring Log Reads and Writes (cont.)

- **Average log read time**
  - Calculated as (`log_read_time / num_log_read_io`)
  - Should be low single digit milliseconds during normal operations (ideally 1 millisecond or less)
  - If higher then storage may be under configured
- **Average log write time**
  - Calculated as (`log_write_time / num_log_write_io`)
  - Should be low single digit milliseconds during normal operations (ideally as low as 1-3 milliseconds, but slightly higher is okay too)
  - If higher than single digits then storage may be under configured

KS-INTERNAL:

Log read and write times are actually returned in seconds and nanoseconds so if implementing one of the formulae then please take into consideration. However, due to a known issue the number of nanoseconds value for log_read_time and log_write_time will always have a value of 000000004 so best just to use the seconds value in any calculations. 05/11/2017: I have no idea if this is still the case or not for 11.1.

I had included a hit ratio formula here in the past but I removed it because it isn't invalid. I discussed this with Roger at some point and he agreed that things didn't look right and it was difficult to get this kind of thing. I could consider using a variation of Scott Hayes' formula if it seems okay, but I haven't had time to validate it:

The Log Read Hit Ratio (LGRHR)

LGRHR = 100 – ((Number read log IOs * 100) / Log pages read)

Time Spent Metrics for Logging

- **Reported through monitor table functions**
  - e.g. `MON_GET_UNIT_OF_WORK` and `MON_GET_WORKLOAD`
- **`log_disk_waits_total`**
  - Number of times agents have to wait for log data to write to disk
  - e.g. Commit processing, flushing logs when writing data page to disk
- **`log_disk_wait_time`**
  - Amount of time an agent spends waiting for log records to be flushed to disk (in milliseconds)
- **`log_buffer_wait_time`**
  - Amount of time an agent spends waiting for space in the log buffer (in milliseconds)
  - If time seems excessive then consider increasing `LOGBUFSZ`
- **`num_log_buffer_full`**
  - Number of times agents are unable to copy records into log buffer because it is full (log buffer is flushed to disk and agents have to wait – as above)

<No speaker notes for this slide>

Example Log Monitor Queries

Q1. How much of the active log space is currently being used?

```
SELECT LOG_UTILIZATION_PERCENT AS "LOGSPC_USED (%)",
       INT(TOTAL_LOG_USED_KB / 1024) AS "LOGSPC_USED (MB)",
       INT(TOTAL_LOG_AVAILABLE_KB / 1024) AS "LOGSPC_FREE (MB)",
       INT(TOTAL_LOG_USED_TOP_KB / 1024) as "MAX_LOGSPC_USED (MB)"
  FROM SYSIBMADM.MON_TRANSACTION_LOG_UTILIZATION;

LOGSPC_USED (%) LOGSPC_USED (MB) LOGSPC_FREE (MB) MAX_LOGSPC_USED (MB)
--------------- ---------------- ---------------- --------------------
          10.52               26              227                  149

  1 record(s) selected.
```


For older versions of DB2 that don't have the SYSIBMADM.MON_TRANSACTION_LOG_UTILIZATION admin view, use the SYSIBMADM.LOG_UTILIZATION admin view instead (same columns can be queried). Another alternative to this query is to use TOTAL_LOG_AVAILABLE, TOTAL_LOG_USED, and TOT_LOG_USED_TOP from the MON_GET_TRANSACTION_LOG table function.

Example Log Monitor Queries (cont.)

Q2. What is the range of the active log files and the current position in them?

```
SELECT FIRST_ACTIVE_LOG, LAST_ACTIVE_LOG,
  CURRENT_ACTIVE_LOG FROM TABLE(MON_GET_TRANSACTION_LOG(NULL)) AS T;

FIRST_ACTIVE_LOG     LAST_ACTIVE_LOG      CURRENT_ACTIVE_LOG
-------------------- -------------------- --------------------
                  15                   40                   35

  1 record(s) selected.
```


For older versions of DB2 that don't have the MON_GET_TRANSACTION_LOG table function, use the SYSIBMADM.SNAPDETAILLOG admin view instead (same columns can be queried).

Example Log Monitor Queries (cont.)

Q3. How much of the current log space is being used, how large is the recovery window, and how much log space is being held up by dirty (modified) pages?

```
SELECT (TOTAL_LOG_AVAILABLE / (1024 * 1024)) AS "TOT_LOG_AVAIL (MB)",
       (TOTAL_LOG_USED / (1024 * 1024)) AS "TOT_LOG USED (MB)",
       (LOG_TO_REDO_FOR_RECOVERY / (1024 * 1024)) AS "LOG_REDO_RECOV (MB)",
       (LOG_HELD_BY_DIRTY_PAGES / (1024 * 1024)) AS "LOG_HELD_BY_DP (MB)"
  FROM TABLE(MON_GET_TRANSACTION_LOG(NULL)) AS T;

TOT_LOG_AVAIL (MB) TOT_LOG USED (MB) LOG_REDO_RECOV (MB) LOG_HELD_BY_DP (MB)
------------------ ---------------- ------------------- --------------------
               229               24                  20                   20

  1 record(s) selected.
```


<No speaker notes for this slide>

Example Log Monitor Queries (cont.)

Q4. What is the oldest write transaction running and how much log space is it consuming?

```
SELECT INT(T.APPLID_HOLDING_OLDEST_XACT) AS OLD_APP_ID,
       CHAR(CONN.SYSTEM_AUTH_ID,10) AS AUTHID,
       CHAR(CONN.APPLICATION_NAME,10) AS APPNAME,
       INT(UOW.UOW_LOG_SPACE_USED) AS LOG_USED_B,
       CHAR(UOW.WORKLOAD_OCCURRENCE_STATE,10) AS STATUS,
       UOW.UOW_START_TIME
  FROM TABLE(MON_GET_TRANSACTION_LOG(NULL)) AS T,
       TABLE(MON_GET_UNIT_OF_WORK(NULL,NULL)) AS UOW,
       TABLE(MON_GET_CONNECTION(NULL,NULL)) AS CONN
  WHERE T.APPLID_HOLDING_OLDEST_XACT = UOW.APPLICATION_HANDLE AND
        UOW.APPLICATION_HANDLE = CONN.APPLICATION_HANDLE;

OLD_APP_ID  AUTHID     APPNAME    LOG_USED_B  STATUS     UOW_START_TIME
----------- ---------- ---------- ----------- ---------- --------------------------
       1340 KSCHLAMB   db2bp.exe        10107 UOWWAIT    2017-05-11-17.31.24.146479

  1 record(s) selected.
```


<No speaker notes for this slide>

Example Log Monitor Queries (cont.)

Q5. Who are the largest consumers of log space running in the database?

```
SELECT CHAR(CONN.SYSTEM_AUTH_ID,10) AS AUTHID,
       CHAR(CONN.APPLICATION_NAME,10) AS APPNAME,
       CHAR(UOW.WORKLOAD_OCCURRENCE_STATE,10) AS STATUS,
       INT(UOW.UOW_LOG_SPACE_USED) AS LOG_USED_B,
       UOW.UOW_START_TIME
 FROM TABLE(MON_GET_UNIT_OF_WORK(NULL,NULL)) AS UOW,
      TABLE(MON_GET_CONNECTION(NULL,NULL)) AS CONN
 WHERE UOW.APPLICATION_HANDLE = CONN.APPLICATION_HANDLE AND
       UOW.UOW_LOG_SPACE_USED > 0
 ORDER BY UOW.UOW_LOG_SPACE_USED DESC;
```

```
AUTHID     APPNAME    STATUS     LOG_USED_B  UOW_START_TIME
---------- ---------- ---------- ----------- --------------------------
KSCHLAMB   db2bp.exe  UOWWAIT        9752606 2017-05-11-17.36.17.286523
KSCHLAMB   db2bp.exe  UOWWAIT        1221084 2017-05-11-17.32.09.090578
KSCHLAMB   db2bp.exe  UOWWAIT          10107 2017-05-11-17.31.24.146479

  3 record(s) selected.
```


For older versions of DB2 that don't have the MON_GET_UNIT_OF_WORK and MON_GET_CONNECTION table functions, use the SYSIBMADM.SNAPAPPL and SYSIBMADM.SNAPAPPL_INFO admin views instead:

```
SELECT CHAR(AI.PRIMARY_AUTH_ID,8) AS AUTH_ID,
    CHAR(AI.APPL_NAME,8) AS APP_NAME,
    CHAR(AI.APPL_STATUS,8) AS STATUS,
    INT(A.UOW_LOG_SPACE_USED) AS LOG_USED_B,
    A.UOW_START_TIME
 FROM SYSIBMADM.SNAPAPPL A,
    SYSIBMADM.SNAPAPPL_INFO AI
 WHERE A.AGENT_ID = AI.AGENT_ID AND
     A.UOW_LOG_SPACE_USED > 0
 ORDER BY A.UOW_LOG_SPACE_USED DESC;
```

Example Log Monitor Queries (cont.)

Q6. What are the average log read and log write times (in milliseconds) for the database?

```
SELECT CASE WHEN NUM_LOG_READ_IO = 0 THEN 0
         ELSE DEC((FLOAT(LOG_READ_TIME) / NUM_LOG_READ_IO),15,3)
       END AS AVG_MS_PER_READ,
       CASE WHEN NUM_LOG_WRITE_IO = 0 THEN 0
         ELSE DEC((FLOAT(LOG_WRITE_TIME) / NUM_LOG_WRITE_IO),15,3)
       END AS AVG_MS_PER_WRITE
  FROM TABLE(MON_GET_TRANSACTION_LOG(NULL)) AS T;

AVG_MS_PER_READ   AVG_MS_PER_WRITE
----------------- -----------------
            0.583             1.537

  1 record(s) selected.
```


For older versions of DB2 that don't have the SYSIBMADM.MON_GET_TRANSACTION_LOG monitor view, use the SYSIBMADM.SNAPDB admin view instead:

```
SELECT CASE WHEN NUM_LOG_READ_IO = 0 THEN 0
        ELSE DEC(((((LOG_READ_TIME_S * 1000000000) + LOG_READ_TIME_NS) /
FLOAT(NUM_LOG_READ_IO)) / 1000000),15,3)
      END AS AVG_MS_PER_READ,
      CASE WHEN NUM_LOG_WRITE_IO = 0 THEN 0
        ELSE DEC(((((LOG_WRITE_TIME_S * 1000000000) + LOG_WRITE_TIME_NS) /
FLOAT(NUM_LOG_WRITE_IO)) / 1000000),15,3)
      END AS AVG_MS_PER_WRITE
  FROM SYSIBMADM.SNAPDB;
```

Example Log Monitor Queries (cont.)

Q7. What are the number of pages read per read I/O and number of pages written per write I/O for the database?

```
SELECT DEC((FLOAT(LOG_WRITES) /
          (NUM_LOG_WRITE_IO + NUM_LOG_PART_PAGE_IO)),15,3)
         AS PAGES_PER_WRITE_IO,
       DEC((FLOAT(LOG_READS) / NUM_LOG_READ_IO),15,3)
         AS PAGES_PER_READ_IO
  FROM TABLE(MON_GET_TRANSACTION_LOG(NULL)) AS T;
```

```
PAGES_PER_WRITE_IO PAGES_PER_READ_IO
------------------ ----------------
            20.723            8.000

  1 record(s) selected.
```


<No speaker notes for this slide>

Example Log Monitor Queries (cont.)

Q8. When needing to read from the logs, what percentage of the time was the data found in the in-memory buffer versus having to go to disk?

```
SELECT NUM_LOG_DATA_FOUND_IN_BUFFER,
       NUM_LOG_READ_IO,
       DEC((FLOAT(NUM_LOG_DATA_FOUND_IN_BUFFER) * 100 /
            (NUM_LOG_READ_IO + NUM_LOG_DATA_FOUND_IN_BUFFER)),15,2)
         AS PERCENT_LOG_DATA_FOUND_IN_BUFFER
  FROM TABLE(MON_GET_TRANSACTION_LOG(NULL)) AS T;

NUM_LOG_DATA_FOUND_IN_BUFFER NUM_LOG_READ_IO    PERCENT_LOG_DATA_FOUND_IN_BUFFER
---------------------------- ----------------- --------------------------------
                      243564              2631                            98.93

  1 record(s) selected.
```


For older versions of DB2 that don't have the SYSIBMADM.MON_GET_TRANSACTION_LOG monitor view, use the SYSIBMADM.SNAPDB admin view instead:

SELECT NUM_LOG_DATA_FOUND_IN_BUFFER, NUM_LOG_READ_IO,

DEC((FLOAT(NUM_LOG_DATA_FOUND_IN_BUFFER) * 100 / (NUM_LOG_READ_IO + NUM_LOG_DATA_FOUND_IN_BUFFER)),15,2) AS PERCENT_LOG_DATA_FOUND_IN_BUFFER

FROM SYSIBMADM.SNAPDB;

Example Log Monitor Queries (cont.)

Q9. How often was the in-memory log buffer full, resulting in the agent waiting for the log data to be flushed to disk?

```
SELECT NUM_LOG_BUFFER_FULL
  FROM TABLE(MON_GET_TRANSACTION_LOG(NULL)) AS T;

NUM_LOG_BUFFER_FULL
--------------------
                   9

  1 record(s) selected.
```


<No speaker notes for this slide>

Further Reading

- **Lots of other topics related to logging and recovery that we didn't have time to get to today**
- **Logging-related sections in the Information Center**
  - Search for:
    - Database Logging
    - Configuring Database Logging Options
    - Configuration Parameters for Database Logging
    - Data Recovery

<No speaker notes for this slide>