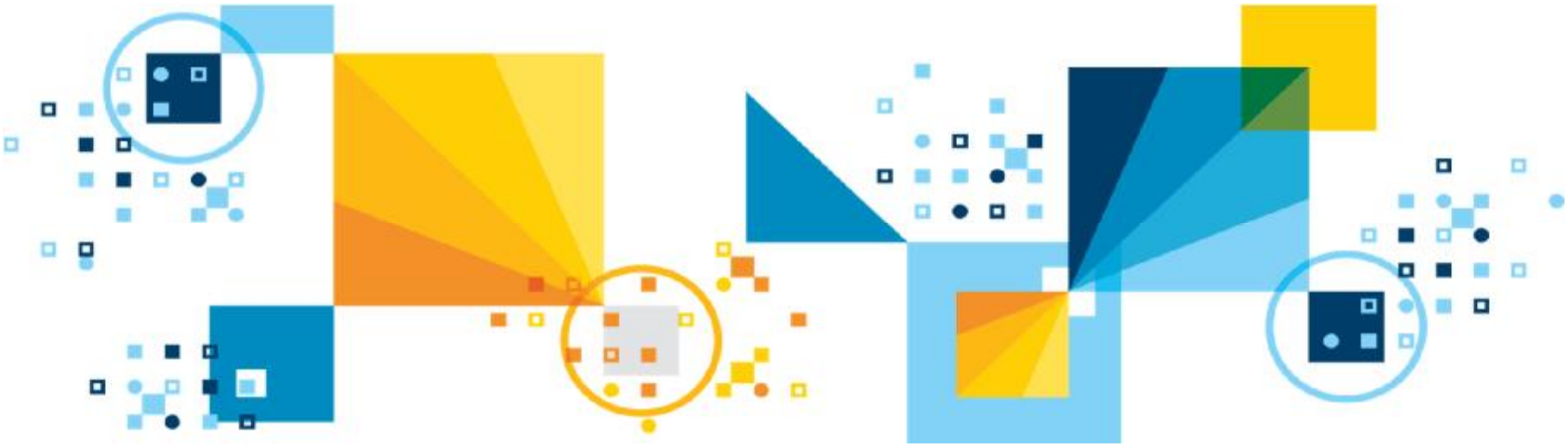
DB2 Locking and Logging for Performance

Module ID	10305
Length	1 hour + 2 hour hands on lab



For questions about this presentation contact askdata@ca.ibm.com

February 9, 2015

PRESENTATION NOTES FOR PAGE 1

Disclaimer

PRESENTATION NOTES FOR PAGE 2

Module Information

- You should have completed or acquired the necessary knowledge for the following modules in order to complete this module:
  - Module DB2 Performance Monitoring Essentials
  - Module DB2 Performance Monitoring Essentials II
  - Module DB2 SQL Query Tuning with BLU acceleration
  - Module DB2 SQL Query Tuning with BLU acceleration II

- After completing this module, you should be able to:
  - Describe the major functions of DB2 locking and logging
  - Able to perform configuration and performance tuning for DB2 locking and logging

PRESENTATION NOTES FOR PAGE 3

Agenda

- Locking and Performance
- Identifying Locking Scenarios
- Using Isolation Levels
- Monitoring Locking Issues
- Avoiding Locking Scenarios

PRESENTATION NOTES FOR PAGE 4

We will start off by discussing Concurrency and factors that affect Concurrency, which will lead into database isolation levels.

We will then look at how DB2 uses locking to control concurrency and enforce isolation levels.

DB2 Monitoring techniques will explain how you can use the tools to identify when and where concurrency problems are occurring.

These Concurrency Problems will then be looked at in more detail and recommendations for resolving the more common problems will be made as well as general tuning advice for improving concurrency. Both DB2 Indexes and Utilities will be looked at, since they can have a direct impact on concurrency.

Finally, things will wind down with a review of the recommendations that have been made through the presentation as well as some useful reference materials and websites.

Agenda

- **Locking and Performance**
- **Identifying Locking Scenarios**
- **Using Isolation Levels**
- **Monitoring Locking Issues**
- **Avoiding Locking Scenarios**

PRESENTATION NOTES FOR PAGE 5

Locking and Performance

- To the users, a locking issue can appear to be a performance issue!
- While users wait, the perception is the database cannot retrieve data fast enough…
- … when in fact, the issue may be queries are encountering:
  - Lock Waits
  - Lock Timeouts
  - Lock Escalations
  - Deadlocks

PRESENTATION NOTES FOR PAGE 6

Locking and concurrency issues can have a significant impact on the performance of a DB2 application. This means the database administrators and application developers needs a good understanding of how the various locking scenario work in DB2.

You may have a locking problem, if…

§ You are experiencing:

- Application failures
- Frequent retries of application processing
- General “slow down” in performance of SQL processing

§ Your objectives:

- Reduce or eliminate all lock timeouts and deadlocks
  - Both result in application failures
  - Extra processing time
  - Wasted system resources
- Eliminate all lock escalations

PRESENTATION NOTES FOR PAGE 7

Monitoring Locking

- **Use snapshots, event monitors, administrative views and db2pd**
  - Key monitoring elements can be retrieved multiple ways
- **Review Administrative Notification logs**
- **Need to know:**
  - Type of lock event causing performance slowdown
  - Identify the SQL statement(s) involved
  - Lock requested and encountered by applications
- **Data collection can be difficult**
  - Locking information is extremely transient
    - Most lock information is gone once the lock is released
  - Baseline data is needed for comparison and evaluation

Recommended to monitor for lock waits, lock timeouts and deadlock events at all times

PRESENTATION NOTES FOR PAGE 8

To confirm what type of locking problem is the reason for your SQL query performance slowdown or query completion failure, it is necessary to collect information that would help to identify what type of lock event is involved, which application is requesting or holding this lock, what was the application doing during this event, and the SQL statement or statements that are involved in being noticeably slow.

The creation of a locking event monitor, use of a table function and views, or use of the db2pd command can collect this type of information.

The information gathered by the locking event monitor can be categorized into three main categories:

Information about the lock in question

Information about the application requesting this lock and its current activities. In the case of a deadlock, this is information about the statement referred to as the victim.

Information about the application owning the lock and its current activities. In the case of a deadlock, this is information about the statement referred to as the participant.

Agenda

- Locking and Performance
- **Identifying Locking Scenarios**
- Using Isolation Levels
- Monitoring Locking Issues
- Avoiding Locking Scenarios

PRESENTATION NOTES FOR PAGE 9

Lock Scenarios

§ Lock waits
- Typical, expected event
  - Excessive time spent waiting for a lock is not typical
  - Lock waits slow down performance and completion
  - Excessive lock waits can become lock timeouts

§ Lock escalations
- Small number acceptable – only if no adverse effects
- Contributes to other locking issues (e.g. lock timeouts)
- Objective should be to eliminate all lock escalations

§ Lock timeout
- Lock timeouts result in the application not completing the transaction and performing a rollback

§ Deadlocks
- Resolution deadlock detector arbitrarily selects one deadlocked process as the victim process to roll back

PRESENTATION NOTES FOR PAGE 10

Lock Wait Behavior. If one application requests to update a row that is already locked with an exclusive (X) lock the application requesting the update will simply wait until the exclusive lock is released by the other application. When an unconditional lock request/upgrade must wait on another transaction (compatibility).

Lock Escalations. If your application changes many rows in one table, it may be better to have one lock on the entire table rather than many locks on each of the rows. DB2 requires memory for each lock; therefore, if a number of row locks can be replaced with a single table lock, the locking storage area can be used by other applications. When DB2 converts the row locks to a table lock on your behalf, this is called lock escalation. DB2 will perform lock escalation to avoid resource problems by too many resources being held for a individual locks.

Lock timeout. A common user symptom of a locking problem is an application hang. A hang can appear as "Lock Wait" within the database engine.

- specifies the number of seconds that an application will wait to obtain a lock

Deadlocks. A deadlock is created when two applications are each locking data needed by the other, resulting in a situation when neither application can continue execution. Because applications do not voluntarily release locks on data that they need, a deadlock detector process is required to break deadlocks and allow application processing to continue.

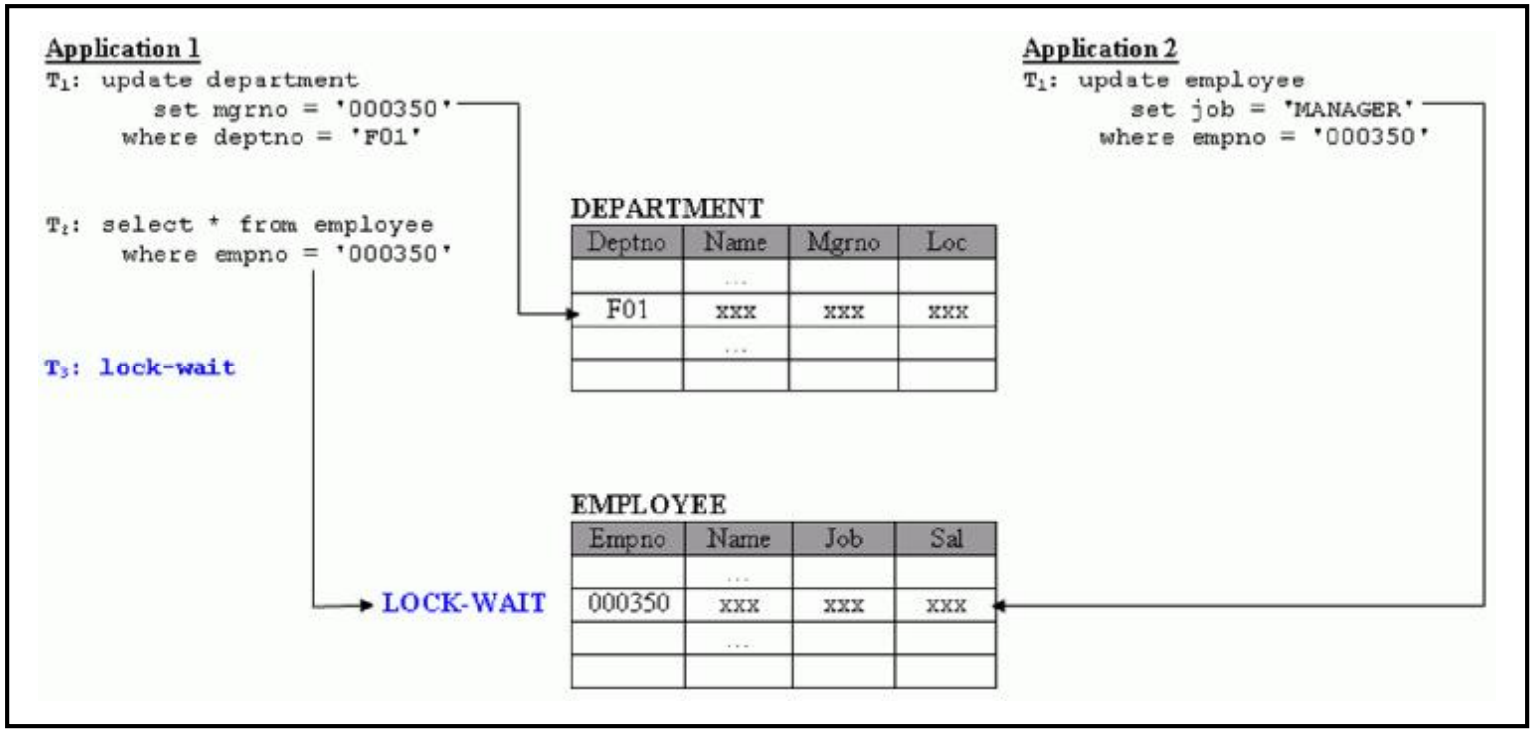
Lock Wait Scenario

§ Application requests a lock whose mode conflicts with lock held by another

§ Requesting application is suspended in a “lock wait” mode until:

– Transaction causing conflict releases lock (i.e., COMMIT, ROLLBACK or FORCE APPLICATION)

– Lock timeout (or deadlock) occurs



PRESENTATION NOTES FOR PAGE 11

A lock-wait occurs when one transaction requests a lock whose mode conflicts (see Lock Compatibility) with the lock held by another transaction. The requesting transaction is suspended and temporarily stops running. The lock-wait will continue until the transaction causing the conflict releases its lock (i.e., COMMIT, rollback, or FORCE APPLICATION) or a lock timeout (or deadlock) occurs. If LOCKTIMEOUT has been set to -1, a transaction will wait indefinitely for a lock and no lock timeout will occur.

Deadlock Scenario – Example – Cursor Stability

Transaction A	Transaction B
update **T1** set col1 = ? where col2 = 2	
	update **T2** set col1 = ? where col2 = ?
select * from **T2** where col2 >= ?	
	select * from **T1** where col5 = ? and col2 = ?

Waiting because is reading uncommitted data

Waiting because is reading uncommitted data

DEADLOCK!! 

PRESENTATION NOTES FOR PAGE 12

Here is an example showing two applications using the isolation level of Read Stability to hold a read lock while a decision to update the row is made by an application user.

The Read locks would allow two users to acquire the data for review at the same time.

If the first user decides to update the row, the exclusive lock needed to perform the update would be delayed by the other user's read lock.

If the second user also decides to update the row, that application would enter a lock wait trying to get the same exclusive lock.

This would cause a deadlock situation that would be resolved when DB2 decides to roll back one of the two applications to allow the other to process normally. The applications would need to be developed to handle this possibility.

Deadlock

- **All deadlocks are considered abnormal**
- **Indicators include processing delays and poor performance**
- **Deadlock slows down the participant transaction while it waits for deadlock detection and resolution**
  - Wastes system resources by rolling back victim transaction
  - Causes extra system work
  - Transaction log access
- **Deadlocks or retry logic in the application cause transactions to be re-executed**
  - The victim application has to re-execute the transaction from the beginning after ROLLBACK

PRESENTATION NOTES FOR PAGE 13

Deadlocks. A deadlock is created when two applications are each locking data needed by the other, resulting in a situation when neither application can continue execution.

We have discussed locking, but what is a deadlock?. A deadlock occurs when two or more applications connected to the same database wait indefinitely for a resource. The waiting can never be resolved because each application is holding a resource that the other needs. As an analogy, in the diagram User A is holding the cereal and needs the milk, User B is holding the milk and needs the cereal to finish making his breakfast. Neither User A nor B is willingly put down what they are currently holding to allow the other to finish, resulting in a deadlock condition.

DB2 Deadlock Detector

§ Responsible for resolving deadlocks

- When a deadlock is detected, the deadlock detector will choose a victim that will be automatically rolled back
  - Rolling back the victim causes the lock conflict to be removed, and the other application can continue processing
  - If it finds a deadlock, the deadlock detector arbitrarily selects one deadlocked process as the victim process to roll back
  - The victim process is awakened, and returns SQLCODE -911 (SQLSTATE 40001), with reason code 2, to the calling application

§ DB CFG parameter DLCHKTIME

- Defines frequency with which the database manager checks for deadlocks
  - A high value increases the deadlock check time and reduces overhead of checking but could result in applications being stuck in deadlock for longer periods of time
  - A low value allows for deadlocks to be detected sooner; however it also introduces additional overhead for checking more frequently

PRESENTATION NOTES FOR PAGE 14

DB2 periodically checks for deadlock situations. The frequency at which it checks for deadlocks is determined by the DLCHKTIME database configuration parameter. If a deadlock is detected, one of the deadlocked applications is chosen as a victim and its unit of work is terminated. This releases the locks it was holding and allows the other transaction to obtain those locks so it can finish its transaction. Setting this parameter to a lower value increases the frequency of deadlock checks, which adds a small performance penalty. Increasing this value to a larger number means that users will potentially wait a longer period to have a deadlock situation resolved, with a lesser performance penalty. You must balance these two criteria to come up with a value that works in your environment.

Deadlocks occur when applications cannot complete a unit of work due to conflicting lock requests that cannot be resolved until the unit of work is completed.

The DB2 deadlock detector is responsible for handling deadlocks

When a deadlock is detected, the deadlock detector will choose a victim that will be automatically rolled back and issued an SQL0911 R.C. 2. By rolling back the victim, the lock conflict is removed, and the other application can continue processing.

The victim is chosen “arbitrarily” DB2 deadlock resolution will return a SQL error code of SQL0911N with a reason code of “2” to the victim process

The frequency at which the DB2 deadlock detector checks for deadlocks can be controlled via DLCHKTIME (measured in milliseconds, from 1000 to 600000). Setting this value high will cause the deadlock check time to be increased but the overhead of deadlock detection will be removed.

This could potentially result in applications stuck in deadlock for prolonged periods of time.

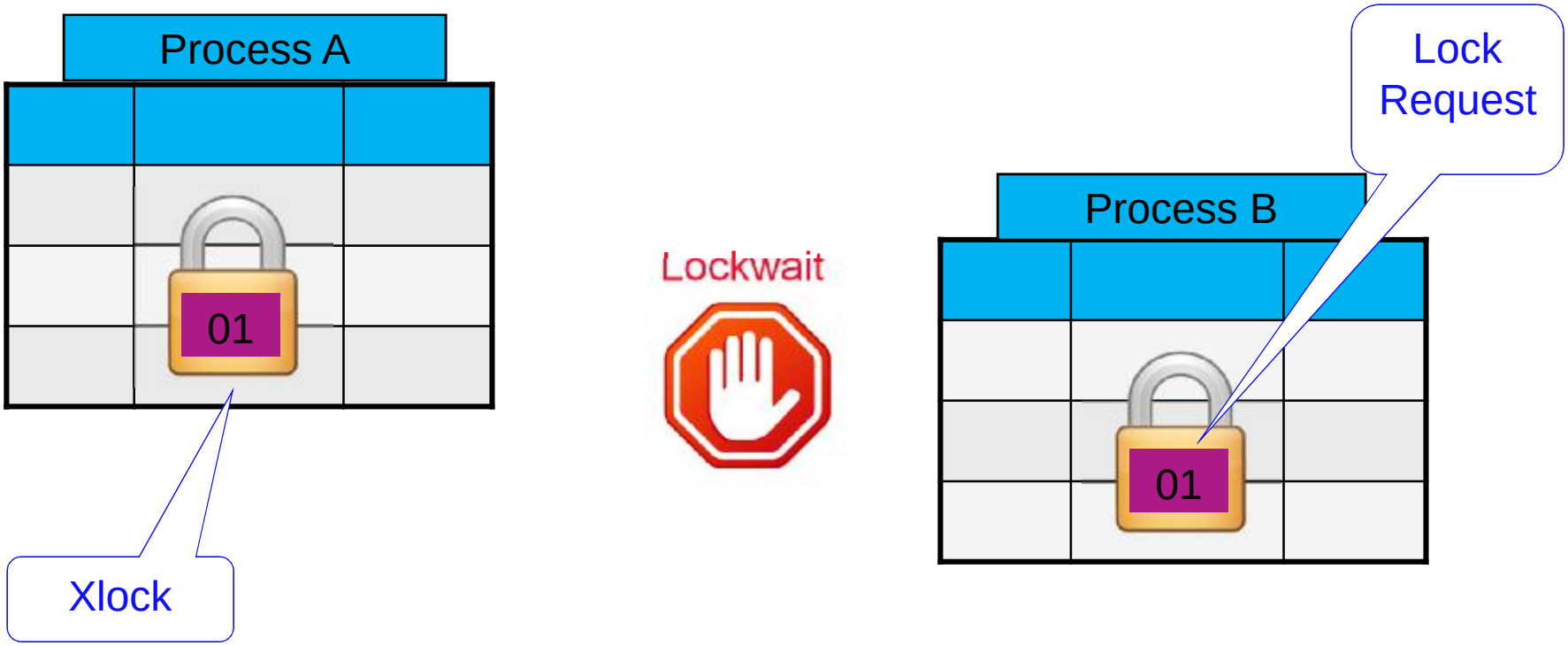
Setting the deadlock check time to a smaller va lue allows for deadlocks to be detected sooner, however it also introduces additional overhead for the checking.

Tracking deadlocks

Snapshots and Snapshot Table Functions provide valuable insight into the number of deadlocks that have occurred

If you find that the number is abnormally large, you should consider enabling an event monitor to track the deadlock events more closely.

Lock Timeout Scenario



DB2 waits for a specified period of time, then returns a SQL error code of SQL0911N with a reason code of “68” to the waiting process

PRESENTATION NOTES FOR PAGE 15

Another situation called a Lock timeout is caused by the unavailability of a resource, can be caused by data being locked by another process
applications that request a lock that is not compatible with the existing locks on the object, or with locks already requested, will be queued.
If the timeout period is exceeded, the waiting application will receive a -911 SQL code.
The unit of work will be automatically rolled back by the database manager

LOCKTIMEOUT and SET CURRENT LOCK TIMEOUT

§ You can set lock wait behavior on a session level rather than using the global value specified by the DB CFG parameter LOCKTIMEOUT

```
          .-CURRENT-.                    .-=-.
>>-SET--+---------+--LOCK TIME OUT--+---+---------------------->

>--+-WAIT-------------------------+-------------------------><
   +-NOT WAIT --------------------+
   +-NULL-------------------------+
   | .-WAIT-.                     |
   +-+------+--integer-constant---+
   '-host-variable-----------------'
```

§ Examples

Example 1: Set the lock timeout value to wait for 30 seconds before returning an error

```
SET CURRENT LOCK TIMEOUT 30
```

Example 2: Unset the lock timeout value, so that the locktimeout database configuration parameter value will be used instead

```
SET CURRENT LOCK TIMEOUT NULL
```

PRESENTATION NOTES FOR PAGE 16

You can customize the lock wait behavior at a session level, rather than using the global value specified by the LOCKTIMEOUT DB CFG parameter. To do this, you can issue the “SET LOCK MODE” statement and indicate “NOT WAIT” to never wait for locks, or specify the amount of seconds to wait using the WAIT n clause, where n is the number of seconds to wait. Using a value of “-1” means that the application will wait indefinitely to obtain the locks it requires.

Lock Escalations

§ Lock escalation can occur in two different scenarios:
- – A single application requests a lock that will exceed its allowable number of locks
- – An application triggers lock escalation because the maximum number of database locks for the entire database is exhausted

§ The database manager will attempt to obtain table locks and release the existing row locks
- – The desired effect is to make more lock memory available for other applications

§ The following database parameters have a direct effect on lock escalation:
- – LOCKLIST - total number of 4k pages allocated for lock storage
- – MAXLOCKS - allowable percentage of locklist that can be used by a single application

§ Tuning and monitoring may be necessary
- – Less of an issue if STMM is managing memory for locks

§ Workload and query behavior dictate locking patterns

PRESENTATION NOTES FOR PAGE 17

Lock escalation from multiple row-level locks to a single table-level lock can occur for the following reasons:

The total amount of memory consumed by many row-level locks held against a table exceeds the percentage of total memory allocated for storing locks

The lock list runs out of space. The application that caused the lock list to be exhausted will have its locks forced through the lock escalation process, even though the application is not the holder of the most locks.

The threshold percentage of total memory allocated for storing locks, that has to be exceeded by an application for a lock escalation to occur, is defined by the maxlocks database configuration parameter and the allocated memory for locks is defined by the locklist database configuration parameter. In a well-configured database, lock escalation is rare. If lock escalation reduces concurrency to an unacceptable level, you can analyze the problem and decide on the best course of action.

The desired effect is to make more lock memory available for additional applications.

Tuning and monitoring may be necessary to find a balance for these values. Workload, and query behavior dictate locking patterns and how applications will use lock memory.

Lock escalation is less of an issue, from the memory space perspective, if self tuning memory manager (STMM) is managing the memory for locks that is otherwise only allocated by the locklist database configuration parameter. STMM will automatically adjust the memory space for locks if it ever runs out of free memory space.

Monitoring/Identifying Locking Issues in General

§ Key indicator monitoring elements:

- lock_timeouts or lock_wait_time values are increasing
- int_rollbacks value is increasing
- int_deadlock_rollbacks shows increase in number of rollbacks due to deadlock events
- Check monitoring element lock_escals for indications that lock escalations may be a contributing factor

§ To help identify locking issues, use:

- Administrative table functions and views
  - DBM CFG for DFT_MON_LOCKS must be ON for snapshot table functions to report accurately
  - MON_GET_LOCKS and MON_GET_APPL_LOCKWAIT table functions
- Application and database lock snapshots
- Lock and deadlock event monitors
- db2pd

§ Check Administration Notification Log for:

- Lock escalations
- Lock waits

PRESENTATION NOTES FOR PAGE 18

Ask like a question here. What kind of lock happens if ___ is ^?

Indicative Signs of Locks

Due to the relatively transient nature of locking events, lock event data is most valuable if collected periodically over time

One or more applications are occasionally re-executing transactions

Reducing Locking Occurrences in General

- **Commit the following actions as soon as possible:**
  - Write actions such as DELETE, INSERT, and UPDATE
  - Data definition language (DDL) statements (e.g. ALTER, CREATE, and DROP statements)
  - BIND and REBIND commands
- **Avoid fetching result sets that are larger than necessary**
  - The more rows that are touched, the more locks that are held, the greater the opportunity to run into a locking problem
  - Push down row selection criteria into a WHERE clause of the SELECT statement
    - As opposed to returning rows and filtering them at the application
- **Avoid using a higher isolation level than necessary**
- **Use WITH RELEASE clause with CLOSE CURSOR statement**

PRESENTATION NOTES FOR PAGE 19

It is best practice to commit the following actions as soon as possible:

Write actions such as delete, insert, and update

Data definition language (DDL) statements, for example ALTER, CREATE, and DROP statements

BIND and REBIND commands

Avoid fetching result sets that are larger than necessary. The more that rows are touched, the more that locks are held, and the greater the opportunity to run into a lock that is held by someone else especially under the repeatable read (RR) isolation level. In practical terms, this often means pushing down row selection criteria into a WHERE clause of the SELECT statement, rather than bringing back more rows and filtering them at the application.

Avoid using higher isolation levels than necessary. e.g. Repeatable read might be necessary to preserve result set integrity in your application; however, it does incur extra cost in terms of locks held and potential lock conflicts.

WITH RELEASE The release of all locks that have been held for the cursor is attempted. Note that not all of the locks are necessarily released; these locks may be held for other operations or activities.

Resolving Deadlock Issues

§ Deadlock frequency can be reduced by ensuring that all applications access common data in the same order

- When two applications take incompatible locks on the same objects in different order, they run a much larger risk of deadlocking

§ Avoid concurrent DDL operations if possible

- For example, DROP TABLE statements can result in a number of catalog updates as rows might have to be deleted for the table indexes, primary keys, check constraints in addition to the table
- If other DDL operations are dropping or creating objects, there can be lock conflicts and even occasional deadlocks

PRESENTATION NOTES FOR PAGE 20

There are a number of guidelines that help to reduce lock contention and lock wait time.

Consider the following options:

Each application connection should process its own set of rows to avoid lock waits.

Deadlock frequency can sometimes be reduced by ensuring that all applications access their common data in the same order meaning, for example, that they access (and therefore lock) rows in Table A, followed by Table B, followed by Table C, and so on. If two applications take incompatible locks on the same objects in different order, they run a much larger risk of deadlocking.

Avoid concurrent DDL operations if possible. For example, DROP TABLE statements can result in a large number of catalog updates as rows might have to be deleted for the table indexes, primary keys, check constraints, etc.

A lock timeout is not much better than a deadlock, because both cause a transaction to be rolled back, but if you must minimize the number of deadlocks, you can do it by ensuring that a lock timeout will usually occur before a potential related deadlock can be detected. To do this, set the value of the locktimeout database configuration parameter (units of seconds) to be much lower than the value of the dlchktime database configuration parameter (units of milliseconds). Otherwise, if locktimeout is longer than the dlchktime interval, the deadlock detector could wake up just after the deadlock situation began, and detect the deadlock before the lock timeout occurs.

Eliminating Lock Escalations

§ Combination of good application design and database configuration can minimize or eliminate lock escalations

- If possible, acquire an explicit table lock with LOCK TABLE statement
  - Minimizes the DB2 workload and reduces the associated system resources needed

§ If not using STMM, manually adjust MAXLOCKS or LOCKLIST

- Their values may be too small for your current workload
  - If multiple applications are experiencing lock escalation, this could be an indication that the LOCKLIST needs to be increased
  - If only one application is experiencing lock escalations, then adjusting MAXLOCKS could resolve this issue

PRESENTATION NOTES FOR PAGE 21

The best technique for resolution is a combination of good application design and database configuration can minimize or eliminate lock escalations.

You can modify the application to acquire table locks using the LOCK TABLE statement.

This is a good strategy for tables where concurrent access by many applications and users is not critical; for example, when the application uses a permanent work table (for example, not a DGTT) that is uniquely named for this instance of the application. Acquiring table locks would be a good strategy in this case as it will reduce the number of locks being held by the application and increase the performance because row locks no longer need to be acquired and released on the rows that are accessed in the work table. If the application does not have work tables and you cannot increase the values for locklist or maxlocks configuration parameters, then you can have the application acquire a table lock. However, care must be taken in choosing the table or tables to lock. Avoid tables that are accessed by many applications and users because locking these tables will lead to concurrency problems which can affect response time, and, in the worst case, can lead to applications experiencing lock timeouts.

Increase the frequency of commits in the application, if business needs and the design of the application allow this. Increasing the frequency of commits reduces the number of locks that are held at any given time. This helps to prevent the application from reaching the maxlocks value, which triggers a lock escalation, and helps to prevent all the applications from exhausting the lock list.

If not using STMM, manually adjust the maxlocks or locklist database configuration parameters

The value of the locklist configuration parameter may be too small for your current workload. If multiple applications are experiencing lock escalation, this could be an indication that the lock list size needs to be increased.

Growth in workloads or the addition of new applications could cause the lock list to be too small. If only one application is experiencing lock escalations, then adjusting the maxlocks configuration parameter could resolve this.

You may want to consider increasing locklist at the same time you increase maxlocks — if one application is allowed to use more of the lock list, all the other applications could now exhaust the remaining locks available in the lock list and experience escalations.

You can modify the application to acquire table locks using the LOCK TABLE statement

This is a good strategy for tables where concurrent access by many applications and users is not critical; for example, when the application uses a permanent work table (for example, not a DGTT) that is uniquely named for this instance of the application. Acquiring table locks would be a good strategy in this case as it will reduce the number of locks being held by the application and increase the performance because row locks no longer need to be acquired and released on the rows that are accessed in the work table. If the application does not have work tables and you cannot increase the values for locklist or maxlocks configuration parameters, then you can have the application acquire a table lock. However, care must be taken in choosing the table or tables to lock. Avoid tables that are accessed by many applications and users because locking these tables will lead to concurrency problems which can affect response time, and, in the worst case, can lead to applications experiencing lock timeouts.

Agenda

- Locking and Performance
- Identifying Locking Scenarios
- **Using Isolation Levels**
- Monitoring Locking Issues
- Avoiding Locking Scenarios

PRESENTATION NOTES FOR PAGE 22

Isolation Levels

§ DB2 provides different levels of protection to isolate data

	Isolation Level	Dirty Read	Non-repeatable Read	Phantom Read	
	Repeatable Read (RR)	-	-	-	
	Read Stability (RS)	-	-	Possible	
DEFAULT	Cursor Stability (CS)	-	Possible	Possible	DEFAULT
	Uncommitted read (UR)	Possible	Possible	Possible	

§ Isolation level can be specified at many levels
- Connection
- Session (application)
- Statement

§ For Embedded SQL, the level is set at bind time

§ For Dynamic SQL, the level is set at run time

PRESENTATION NOTES FOR PAGE 23

DATA INTEGRITY REQUIREMENTS

DB2 provides different levels of isolation to control how data is accessed and changed. The isolation level chosen influences the type and duration of locks obtained by DB2 on behalf of the application in the course of a transaction. There are four supported levels of isolation, “Uncommitted Read”, “Cursor Stability”, “Read Stability” and “Repeatable Read”.

The isolation level is usually specified by the application connecting to the database. If no isolation level is specified by the application, DB2 will automatically use an isolation level of “Cursor Stability”.

Isolation can be set at various levels. It can be specified at a connection level, at a session level, or at a statement level. For Embedded SQL, it is set at bind time, that is, when all the SQL statements are pre-compiled and their associated access plans are registered in the database. For Dynamic SQL, the isolation level is set at run time.

Concurrency Control

§ **Currently Committed** is a variation on DB2's Cursor Stability isolation
- If uncommitted row-change found, use currently committed version of data
- Avoids timeouts and deadlocks
- Log based:
  - No management overhead

Cursor Stability

Situation	Result
Reader blocks Reader	No
Reader blocks Writer	No
Writer blocks Reader	Yes
Writer blocks Writer	Yes

Currently Committed

Situation	Result
Reader blocks Reader	No
Reader blocks Writer	No
Writer blocks Reader	No
Writer blocks Writer	Yes

PRESENTATION NOTES FOR PAGE 24

With Currently Committed behavior, DB2 will not block on locked rows being modified, but instead will access and process the “currently committed” version of the row instead.

Under the new currently committed semantics, only committed data is returned, as was the case previously, but now readers do not wait for updaters to release row locks. Instead, readers return data that is based on the currently committed version; that is, data prior to the start of the write operation.

DB2 is now enabled with full lock avoidance techniques, allowing scans to avoid row locking when the data and/or page are known to be committed.

When the scan cannot determine if an index/row entry is committed or not, traditional conditional locking will take place.

New feedback is encoded in the row lock to identify the log record which (first) modified the row, and is returned by the lock manager on a lock conflict

A Currently Committed scan will use this feedback to access the currently committed version of the row from the log (memory or active log files) to be processed

Uncommitted inserted rows are identified directly in the row lock, allowing CC scanners to immediately ignore and skip the row

FOR SPEAKER’s INFORMATION ONLY:

-> Currently committed semantics require increased log space. Additional space is required for logging the first update of a data row during a transaction. This data is required for retrieving the currently committed image of the row. Depending on the workload, this can have an insignificant or substantial impact on the total log space used. The requirement for additional log space does not apply when cur_commit is disabled.

-> CC is based on data available on the logs. DB2 first goes to the log buffer to find the data. Since the updating transaction is still active, probably the data is still in the log buffers. Otherwise, it goes to the log files on disk. It does not need to search through the log files. DB2 is able to directly access the appropriate log file and the record inside that file (direct I/O access instead of scanning through all log records).

-> In the case it can’t find the record in the log files and INFINITE LOG/log archiving is in place, DB2 switches to CS semantics, ie, readers wait till writer is committed.

Currently Committed – How does it work?

Transaction A	Transaction B
update **T1** set col1 = ? where col2 = 2	
	update **T2** set col1 = ? where col2 = ?
select * from **T2** where col2 >= ?	
	select * from **T1** where col5 = ? and col2 = ?
commit	commit

No locking
Reads last committed version of the data

No locking
Reads last committed version of the data

No deadlocks, no timeouts in this scenario!

PRESENTATION NOTES FOR PAGE 25

Considerations for CUR_COMMIT

- **For new databases, the default is set to ON**
  - When the default is set to ON your query will return the currently committed value of the data at the time when your query is submitted

- During database upgrade from V9.5 or earlier, the cur_commit configuration parameter is set to **DISABLED** to maintain the same behavior as in previous releases

- If you want to use currently committed on cursor stability scans, you need to set the cur_commit configuration parameter to ON after the upgrade

PRESENTATION NOTES FOR PAGE 26

Note: Three registry variables DB2_EVALUNCOMMITTED, DB2_SKIPDELETED, and DB2_SKIPINSERTED are affected by currently committed when cursor stability isolation level is used. These registry variables are ignored when USE CURRENTLY COMMITTED or WAIT FOR OUTCOME are specified explicitly on the BIND or at statement prepare time.

Note: Performance considerations may be applicable in a database where there are significant lock conflicts when using currently committed. The committed version of the row is retrieved from the log, and will perform better and avoid log disk activity when the log record is still in the log buffer. Therefore, to improve the performance of retrieving previously committed data, you might consider an increase to the value of the logbufsz parameter.

Agenda

- Locking and Performance
- Identifying Locking Scenarios
- Using Isolation Levels
- **Monitoring Locking Issues**
- Avoiding Locking Scenarios

PRESENTATION NOTES FOR PAGE 27

Monitoring Locking Issues in General

§ Monitoring:
- Create/configure/enable locking event monitors to capture details on lock event data for a workload or database
  - Find the application that is waiting, and information on the lock requested
- Use db2pd –locks (wait)
- Review DB2 administration notification log (<instance>.nfy) with DIAGLEVEL set to 3 or 4 (waits), or DB2 diagnostic log (db2diag.log)

NEW IN DB2 10

New event monitor features:
- All event monitors now support the WRITE TO TABLE target
- Existing event monitors that write to tables can be altered to capture additional logical data groups

PRESENTATION NOTES FOR PAGE 28

NEW IN DB2 10

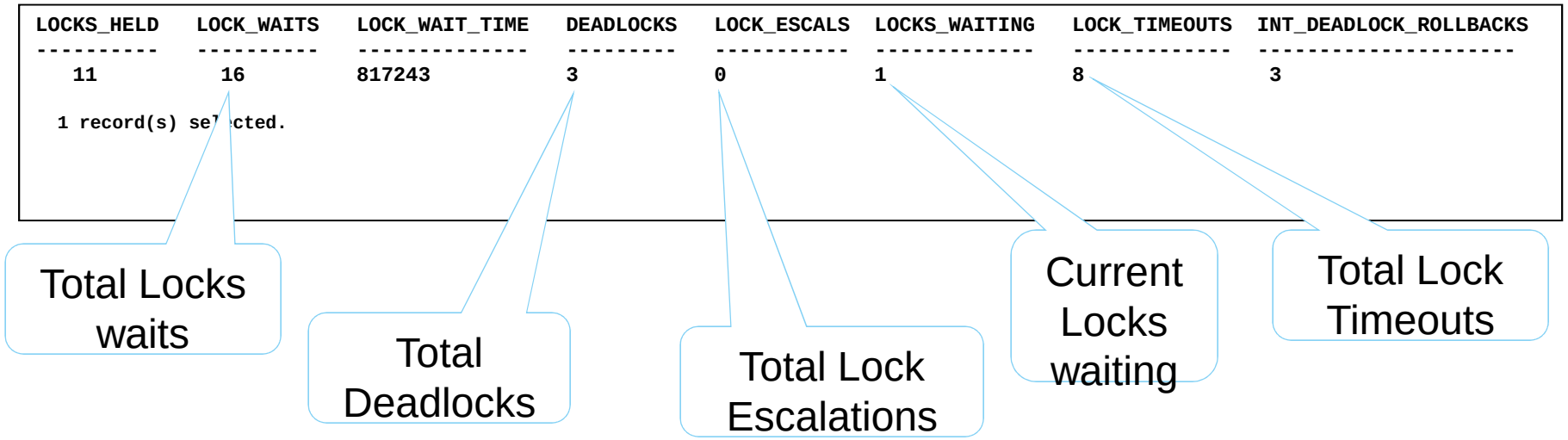
All event monitors now support the WRITE TO TABLE target
http://publib.boulder.ibm.com/infocenter/db2luw/v10r1/topic/com.ibm.db2.luw.wn.doc/doc/c0058910.html

Existing event monitors that write to tables can be altered to capture additional logical data groups.
http://publib.boulder.ibm.com/infocenter/db2luw/v10r1/topic/com.ibm.db2.luw.wn.doc/doc/c0059153.html

Administrative View – SYSIBMADM.SNAPDB

§ **SYSIBMADM.SNAPDB** contains relevant information about locks

```
SELECT  LOCKS_HELD, LOCK_WAITS, LOCK_WAIT_TIME,DEADLOCKS, LOCK_ESCALS,
        LOCKS_WAITING,LOCK_TIMEOUTS, INT_DEADLOCK_ROLLBACKS
FROM SYSIBMADM.SNAPDB;
```



PRESENTATION NOTES FOR PAGE 29

The snapshot elements LOCK_WAIT_TIME and LOCKS_WAITING, available through SYSIBMADM.SNAPDB, indicate total lock wait time and number of agents currently waiting on locks, respectively. A high percentage of active agents waiting on locks (for example, 20% or higher) or an increasing value of lock wait time indicates that locking might be a significant bottleneck.

Administrative View – SYSIBMADM.MON_LOCKWAITS

§ **SYSIBMADM.MON_LOCKWAITS** contains information on locks

```
SELECT SUBSTR(TABSCHEMA,1,8) AS TABSCHEMA
     , SUBSTR(TABNAME,1,15)  AS TABNAME
     , LOCK_OBJECT_TYPE
     , LOCK_MODE
     , LOCK_MODE_REQUESTED
     , AGENT_ID_HOLDING_LK
  FROM SYSIBMADM.MON_LOCKWAITS;
```

```
TABSCHEMA TABNAME         LOCK_OBJECT_TYPE   LOCK_MODE  LOCK_MODE_REQUESTED AGENT_ID_HOLDING_LK
--------- --------------- ------------------ ---------- ------------------- -----------------
DB2INST1  DEPARTMENT      TABLE_LOCK         X          S                                40

  1 record(s) selected.
```

Lock Mode

Lock Mode Requested

SYSIBMADM.LOCKWAITS administrative view is deprecated in DB2 10. Use new view like SYSIBMADM.MON_LOCKWAITS

PRESENTATION NOTES FOR PAGE 30

The LOCKWAITS administrative view returns information about DB2 agents working on behalf of applications that are waiting to obtain locks.

This administrative view is deprecated in DB2 10. You can use a new routine or view:
MON_GET_APPL_LOCKWAIT table function - Get information about locks for which an application is waiting Version 9.7 Fix Pack 1
MON_GET_LOCKS table function - List all locks in the currently connected database
MON_FORMAT_LOCK_NAME table function - Format the internal lock name and return details
MON_LOCKWAITS administrative view - Retrieve metrics for applications that are waiting to obtain locks

Reference: http://publib.boulder.ibm.com/infocenter/db2luw/v10r1/topic/com.ibm.db2.luw.sql.rtn.doc/doc/r0023171.html

Table Functions – MON_GET_LOCKS and MON_GET_APPL_LOCKWAIT

§ Used to investigate locking problems in the current connected database

1. Call the MON_GET_APPL_LOCKWAIT table function to determine all the locks that are waiting to be acquired in the database

```
SELECT lock_name, hld_member, lock_status, hld_application_handle
FROM TABLE (MON_GET_APPL_LOCKWAIT(NULL, -2))
```

§ This query returns the following output:

```
LOCK_NAME                  HLD_MEMBER LOCK_STATUS HLD_APPLICATION_HANDLE
-------------------------- ---------- ----------- --------------------
00030005000000000280000452 -2           W
00030005000000000280000452 -2           W
00030005000000000280000452 -2           W

  3 record(s) selected.
```

Lock status is waiting

§ A HLD_MEMBER value of **-2** indicates that the lock **0x00030005000000000280000452** is being held at a remote member

PRESENTATION NOTES FOR PAGE 31

MON_GET_LOCKS table function - List all locks in the currently connected database

The MON_GET_LOCKS table function returns a list of all locks in the currently connected database.

To get information about locks, use the MON_GET_LOCKS, MON_FORMAT_LOCK_NAME, and MON_GET_APPL_LOCKWAIT table functions, and the MON_LOCKWAIT administrative view instead of the SNAPLOCKWAIT administrative view and SNAP_GET_LOCKWAIT table function, the SNAPLOCK administrative view and SNAP_GET_LOCK table function, and the LOCKS_HELD administrative view which are deprecated in Fix Pack 1 of Version 9.7.

>>-MON_GET_LOCKS--(--search_args--,--member--)----------><

The schema is SYSPROC.

Table function parameters

search_args

An input parameter of type CLOB(1K) that represents a list of key-value pairs. If the list is empty or NULL, all locks in the currently connected database are returned. Otherwise, all locks that match all of the conditions represented by the list of key-value pairs are returned. A key-value pair must follow this format:

A key is a string that consists of an opening tag, followed by the value, followed by a closing tag.

An opening tag consists of an opening angle bracket, followed by the key name, followed by a closing angle bracket. No spaces are allowed.

A closing tag consists of an opening angle bracket, followed by a forward slash, followed by the key name, followed by a closing angle bracket. No spaces are allowed.

All keys are case-sensitive and can only be specified once in the search_args parameter.

The order of the keys does not matter.

SQLCODE -171 is returned for an invalid key-value pair.

SQLCODE -204 is returned if the table does not exist.

MON_GET_LOCKS and MON_GET_APPL_LOCKWAIT (Cont.)

2. Call the MON_GET_LOCKS table function to determine the holder of the lock, by specifying the lock name, 0x00030005000000000280000452, as the search argument:

```
SELECT lock_name, member, lock_status, application_handle
FROM TABLE (MON_GET_LOCKS
( CLOB('<lock_name>00030005000000000280000452</lock_name>'), -2))
```

§ This query returns the following output:

```
LOCK_NAME                          MEMBER LOCK_STATUS APPLICATION_HANDLE
---------------------------------- ------ ----------- ------------------
00030005000000000280000452 0      W               12562
00030005000000000280000452 1      W               12562
00030005000000000280000452 2      G               65545
00030005000000000280000452 3      W               12562

  4 record(s) selected.
```

Lock status is Granted

The App “12562” is waiting to obtain a lock that the App “65545” has

PRESENTATION NOTES FOR PAGE 32

Example

In this sample scenario, the MON_GET_LOCKS and MON_GET_APPL_LOCKWAIT table functions are used to investigate the locking situation in the current connected database, on all members.

1. Call the MON_GET_APPL_LOCKWAIT table function to determine all the locks that are waiting to be acquired in the current connected database, on all members:

The records that show HLD_MEMBER is -2 indicate that the lock 0x00030005000000000280000452 is being held at a remote member.

2. Call the MON_GET_LOCKS table function to determine the holder of the lock, by specifying the lock name, 0x00030005000000000280000452, as the search argument:

To find out more about the application holding the lock, you can call the WLM_GET_SERVICE_CLASS_WORKLOAD_OCCURRENCES or WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES table functions.

Monitoring db2diag.log for Lock Waits

§ DBM CFG parameter DIAGLEVEL set to 4 records lock timeouts

```
2009-09-04-10.16.41.126755-240 E5543901G631   LEVEL: Info
PID     : 6974           TID  : 2950687648  PROC : db2sysc 0
INSTANCE: db2inst1       NODE : 000         DB   : SAMPLE
...
...
...
Request for lock "TAB: (2, 6)" in mode ".IX" timed out
Application caused the lock wait is
"*LOCAL.db2inst1.090904141554"
Statement:
DATA #2 : String with size, 41 bytes
update employee set edlevel = edlevel + 1
```

Attempt to acquire a table lock

Lock requested Intent exclusive (IX)

Statement causing the error

PRESENTATION NOTES FOR PAGE 33

DIAGLEVEL 4 logs more information than lower levels. At this level (4) additional information concerning locking can be captured in the db2diag.log.

Do not set it at that level during normal daily activities; use it only as a troubleshooting aid.

Monitoring database locks using db2pd

```
db2pd –db sample –locks wait
```

```
Database Partition 0 -- Database SAMPLE -- Active -- Up 0 days 07:00:28

Locks:
Address    TranHdl    Type       Mode Sts Owner    Dur HoldCount  Att  ReleaseFlg
0xA5AB63F0  8         Row         ..S  W     9      1      0      0x10 0x00000002
0xA5AB61E0  9         Row         ..X  G     9      1      0      0x00 0x40000000
```

Lock Level

Lock Mode (exclusive)

Lock Status "Waiting"

PRESENTATION NOTES FOR PAGE 34

will show you each of the locks in a wait status and their associated waiter.

“Status”: A value of “Lock-wait” means the application is blocked by a lock held by a different application. Don’t be confused by a value of “UOW Waiting”: it means that the application (unit of work) is in progress, not blocked on a lock, but is just not doing any work at the moment. However, when a lock conflict does occur, an application holding locks that others are blocked on will usually have a status of “UOW Waiting”, so that status is not necessarily an innocuous one.

“Status Change Time”: This is of particular interest for an application with Lockwait status: it shows when the lock wait began. Note that the UOW monitor switch must be on for this time to be reported.

“Appl. Handle”: The handle is an integer value that serves two main purposes: It allows the LIST APPLICATIONS information to be correlated with the output of snapshot and event monitors discussed below.

It provides the value that you can use in the FORCE APPLICATION command to force an application that’s holding locks which are causing contention problems.

Various other columns identify the connection and application, including the database name.

Monitoring - Lock Escalation

§ With DBM CFG parameter DIAGLEVEL set to 3 (default) or 4, Lock escalations are reported in the db2diag.log:

```
2009-07-13-16.27.49.115000-240 E1444H457          LEVEL: Warning
PID     : 2800                 TID  : 3444        PROC : db2syscs.exe
INSTANCE: DB2                  NODE : 000         DB   : SAMPLE
APPHDL  : 0-146                APPID: *LOCAL.DB2.050713222002
FUNCTION: DB2 UDB, data management, sqldEscalateLocks, probe:3
MESSAGE : ADM5502W  The escalation of "240671" locks on table
"DB2ADMIN.EMPLOYEE"
          to lock intent "X" was successful.
```

§ Additional tools

– IBM Data Studio & IBM InfoSphere Optim Performance Manager
– Snapshot Monitor (Database and Application level) and Event Monitor (Database and Connection type) => "Lock escalations" and "Exclusive lock escalations“ (**x_lock_escals**) info
– Many others DB2 interfaces provide info on "Lock escalations" (**lock_escals**)

In general Control Center and related components as Health Center have been discontinued in DB2 10. These have been replaced by a new set of GUI tools: IBM Data Studio and IBM InfoSphere Optim tools

PRESENTATION NOTES FOR PAGE 35

If the database manager configuration parameter DIAGLEVEL has been set to 3 (default) or 4, when a Lock Escalation occurs, an entry will be made to the db2diag.log. The above shows the data collected with a DIAGLEVEL of 3.

You can also use the following monitors to identify if lock escalations are occurring:

- IBM Data Studio & IBM InfoSphere Optim Performance Manager
- Snapshot Monitor: Database and Application Snapshots' "Lock escalations" and "Exclusive lock escalations"
- Event Monitor: Connections' "Lock escalations" and Databases' "Lock escalations", "X lock escalations"

The number of lock escalations can be identified in Database, Application, and Lock Snapshots.

Connection, Database, and Deadlock Event Monitors also report the number of escalations and can also be useful for tracking the statements which lead up to the lock escalation.

In general, simply knowing that escalations are occurring as well as the table and locks involved should be enough information to resolve the problem. As discussed, this information is found in the db2diag.log.

Agenda

- Locking and Performance
- Identifying Locking Scenarios
- Using Isolation Levels
- Monitoring Locking Issues
- **Avoiding Locking Scenarios**

PRESENTATION NOTES FOR PAGE 36

Avoiding Locking Scenarios

§ Best Practices – Application

- Use least restrictive isolation level that maintains the data integrity requirements of the application
- Reduce Isolation level of specific statements by using statement level isolation (i.e., WITH clause)
- CLOSE cursors WITH RELEASE to free locks prior to end of transaction
- Perform updates as close to the end of the transaction as possible, to reduce exclusive lock duration
- COMMIT frequently to release locks
- Avoid multiple applications accessing the same tables, but acquiring locks in different orders (access patterns should be similar)
- Avoid having multiple processes that access the same table for both reads and writes within the same transaction

PRESENTATION NOTES FOR PAGE 37

8. Recommendation Summary

The following is a quick summary of the recommendations that have been made.

Application

Applications should be designed to balance concurrency with low processing cost. Ideally, provide maximum concurrency while minimizing lock waits, lock timeouts, deadlocks and lock processing overhead, by holding the smallest possible lock on the least number of rows and holding it for the shortest possible duration. The following recommendations, as discussed earlier, can help you with achieving this:

- There is a trade-off between concurrency and lock processing cost: Greatest concurrency with smallest lock on least number of rows held for shortest possible duration; however, this can incur significant CPU processing and memory cost. Table locks can save read-only processing CPU costs.
- Use the least restrictive isolation level that maintains the data integrity requirements of the application, as the less restrictive, the better the data concurrency
- The isolation level used can be redefined at the statement level, allowing you opportunities to further minimize locking
- Use LOCK TABLE SHARE for any read-only tables to reduce locking resources
- Use LOCK TABLE EXCLUSIVE for any batch UPDATE tables which are only accessed by a single application
- Use SELECT FOR READ ONLY on read-only queries to ensure that only share locks are acquired
- Use WHERE CURRENT OF cursor for UPDATE and DELETE to reduce locking scope
- CLOSE cursors WITH RELEASE to free locks prior to end of transaction
- COMMIT frequently to release locks
- To avoid lock escalations, reduce isolation level and COMMIT frequently
- To avoid deadlock:
    - o Specify SELECT…FOR UPDATE if you intend to read and modify the same row within the same transaction. This also minimizes lock conversion costs. Similarly, specify SELECT…WITH RR USE AND KEEP UPDATE LOCKS
    - o Avoid using Explicit Exclusive Table locks, when system-generated row locks would have been sufficient
    - o Avoid having applications which are accessing the same tables, acquiring locks in different orders. Access patterns should be similar.
    - o Avoid having application with several processes that access the same table for both reads then writes within the same transaction
    - o Avoid using an inappropriate/too restrictive isolation level
    - o Be aware that DB2 Catalog tables are locked when using RR
    - o Avoid using ambiguous locking
- Application retry logic should be used to handle ROLLBACK caused by lock timeout or deadlock
- Try to perform updates as close to the end of the transaction as possible, to reduce exclusive lock duration
- Perform batch UPDATE activity during off-peak OLTP periods

Avoiding Locking Scenarios

§ Best Practices – Database

- Avoid lock escalations by increasing DB CFG parameters LOCKLIST and/or MAXLOCKS, or using STMM
- Avoid lock timeouts:
  - Adjust the DB CFG parameter LOCKTIMEOUT or use the SET CURRENT LOCK TIMEOUT command
- Use CURRENTLY COMMITTED
- If not using CURRENTLY COMMITTED, avoid deadlocks by:
  - Reducing row blocking during index and table scans (CS and RS only):
    DB2_SKIPINSERTED to skip/ignore uncommitted inserted rows
    DB2_SKIPDELETED to skip/ignore uncommitted deleted rows
    DB2_EVALUNCOMMITTED to defer locking until row is known to satisfy query. Uncommitted data will be evaluated. Skips deleted rows on table scans

PRESENTATION NOTES FOR PAGE 38

Database

The following are changes which can be made to the database configuration to help you achieve greatest concurrency:

- Make sure that the database manager configuration parameters MAX_CONNECTIONS, MAX_COORDAGENTS, MAXAGENTS, and MAXCAGENTS are sized large enough to allow expected application concurrency and that you have the system resources to support
- Ensure that the database configuration parameter MAXAPPLS is large enough to support expected database application concurrency
- For short running, high volume web applications, the Connection Concentrator can be enabled by setting MAX_CONNECTIONS > MAX_COORDAGENTS
- Avoid lock escalations by increasing the database configuration parameters LOCKLIST and/or MAXLOCKS
- To avoid lock timeouts:
    - o adjust the database configuration parameter LOCKTIMEOUT or use the SET CURRENT LOCK TIMEOUT command
    - o Create indexes on foreign keys to reduce table scans
    - o Reduce row blocking during index and table scans
- To avoid deadlocks:
    - o Eliminate lock escalations
    - o Avoid table scans where index scans are possible
    - o Create indexes on foreign keys to reduce table scans
    - o Reduce row blocking during index and table scans:
        - DB2_SKIPINSERTED to skip/ignore uncommitted inserted rows
        - DB2_SKIPDELETED to skip/ignore uncommitted deleted rows
        - DB2_EVALUNCOMMITTED to defer locking until row is known to satisfy query. Uncommitted data will be evaluated. Skips deleted rows on table scans.

More Useful Registry Variables for Locking

§ DB2_KEEPTABLELOCK

- If set to **`ON`** or **`TRANSACTION`**, this variable allows the DB2 database system to maintain the table lock when an Uncommitted Read or Cursor Stability isolation level is closed. The table lock that is kept is released at the end of the transaction. If set to **`CONNECTION`**, a table lock is released for an application until the application either rolls back the transaction or the connection is reset

§ DB2_MAX_NON_TABLE_LOCKS

- Defines the maximum number of NON table locks a transaction can have before it releases these locks. Because transactions often access the same table more than once, retaining locks and changing their state to NON can improve performance

§ DB2LOCK_TO_RB

- Specifies whether lock timeouts cause the entire transaction to be rolled back, or only the current statement

PRESENTATION NOTES FOR PAGE 39

DB2_KEEPTABLELOCK – Specifies whether the DB2 database system is to maintain table locks when an Uncommitted Read or Cursor Stability isolation level scan is closed. The table lock that is kept is released at the end of the transaction, just as it would be released for Read Stability and Repeatable Read scans.

DB2_MAX_NON_TABLE_LOCKS – Specifies the maximum number of NON table locks a transaction can have before it releases all of these locks. NON table locks are table locks that are kept in the hash table and transaction chain even when the transaction has finished using them. Because transactions often access locks and changing their state to NON can improve performance.

DB2LOCK_TO_RB - Specifies whether lock timeouts cause the entire transaction to be rolled back, or only the current statement. If DB2LOCK_TO_RB is set to STATEMENT, locked timeouts cause only the current statement to be rolled back. Any other setting results in transaction rollback.

Random ordering for Index columns: New in 10.5

- Lessens index page contention
- When rows are added to a table in Index sequence contention on the leave page might occur
- For example keys generated using identity column or sequences or timestamps
- Most beneficial in pureScale environments

PRESENTATION NOTES FOR PAGE 40

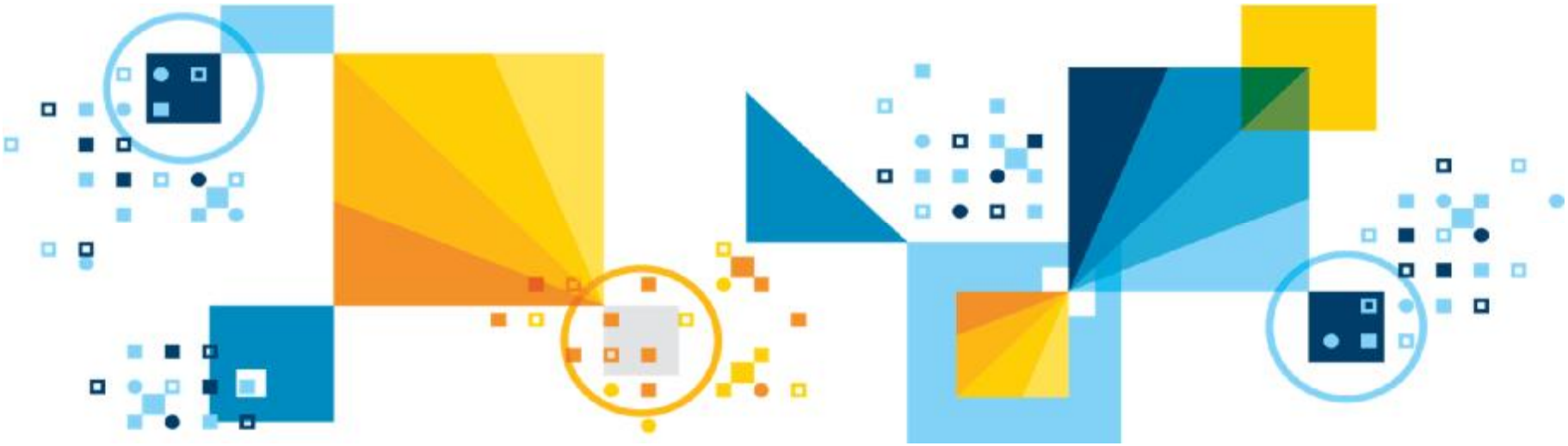
Summary

§ In this Module you learned about:

- Locking and concurrency issues can have a significant impact on the performance of a DB2 application
- It is necessary to collect information that would help to identify what type of lock event is involved
- Commit DML (insert, update, delete) and DDL actions as soon as possible
- Avoid concurrent DDL operations if possible
- Avoid using higher isolation levels than necessary
- Use DB2 options for monitoring locking

PRESENTATION NOTES FOR PAGE 41

DB2 Performance Clinic – Logging



February 9, 2015

PRESENTATION NOTES FOR PAGE 42

DB2 Performance Clinic Modules - Agenda

- **Logging Concepts**
- **Configuration and Performance**
- **Logging Bottlenecks**
- **Reducing Logging**
- **Monitoring and Tuning**

PRESENTATION NOTES FOR PAGE 43

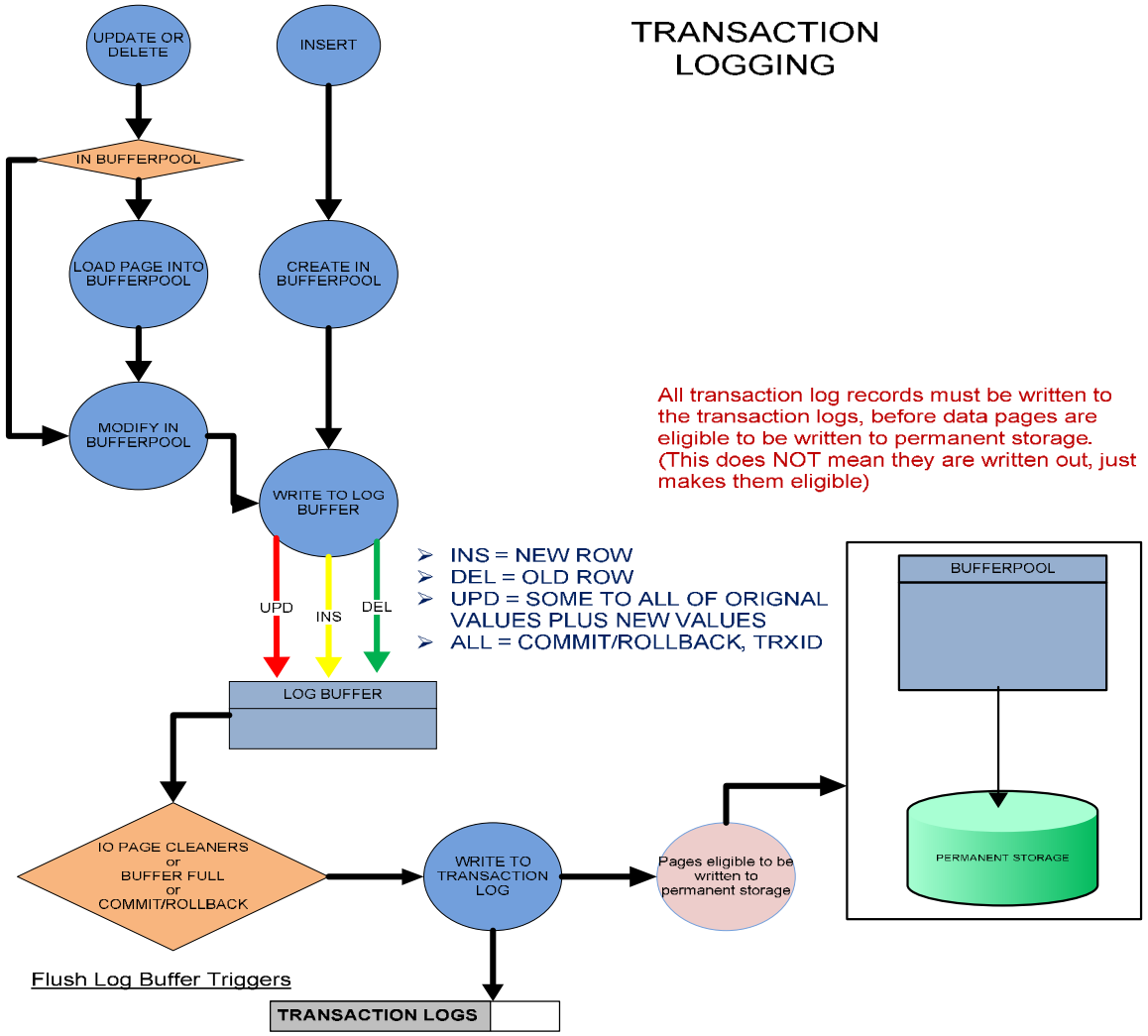
All DB2 databases have logs associated with them. These logs keep a record of all changes made to database objects and data. All the changes are first written to log buffers in memory before being flushed to the log files on disk at COMMIT time. With that said, we will start off by discussing the general logging concepts and then discuss some of the more important database configuration parameters that impact transaction logging. We will then look some of the common logging bottlenecks that can cause the database to appear slow and non-responsive. Finally we will discuss monitoring and tuning the logs.

Agenda

- **Logging Concepts**
- **Configuration and Performance**
- **Logging Bottlenecks**
- **Reducing Logging**
- **Monitoring and Tuning**

PRESENTATION NOTES FOR PAGE 44

Transaction Logging Overview



PRESENTATION NOTES FOR PAGE 45

Before we start discussing transaction logging performance, let's consider the general process of transaction logging. Pages subject to updates, inserts and deletes are manifested in the bufferpools, following that a transaction log record is created and moved into the log buffer. The content and extent of the transaction log record is dependent on what kind of operation is being performed. Inserts, of course will have values for the new row and deletes values for the old row. Updates, specifically, when more than one column is being updated can consist of values from some to all of the columns. Both old and new values need to be tracked. All transaction log records will indicate whether the transaction is rolled back or committed plus the assigned transaction ID.

The log buffer is typically being written to disk pretty much all the time on most systems. But there are three things that trigger this action

Any time the I/O page cleaners are triggered to write pages in the bufferpool to permanent storage.

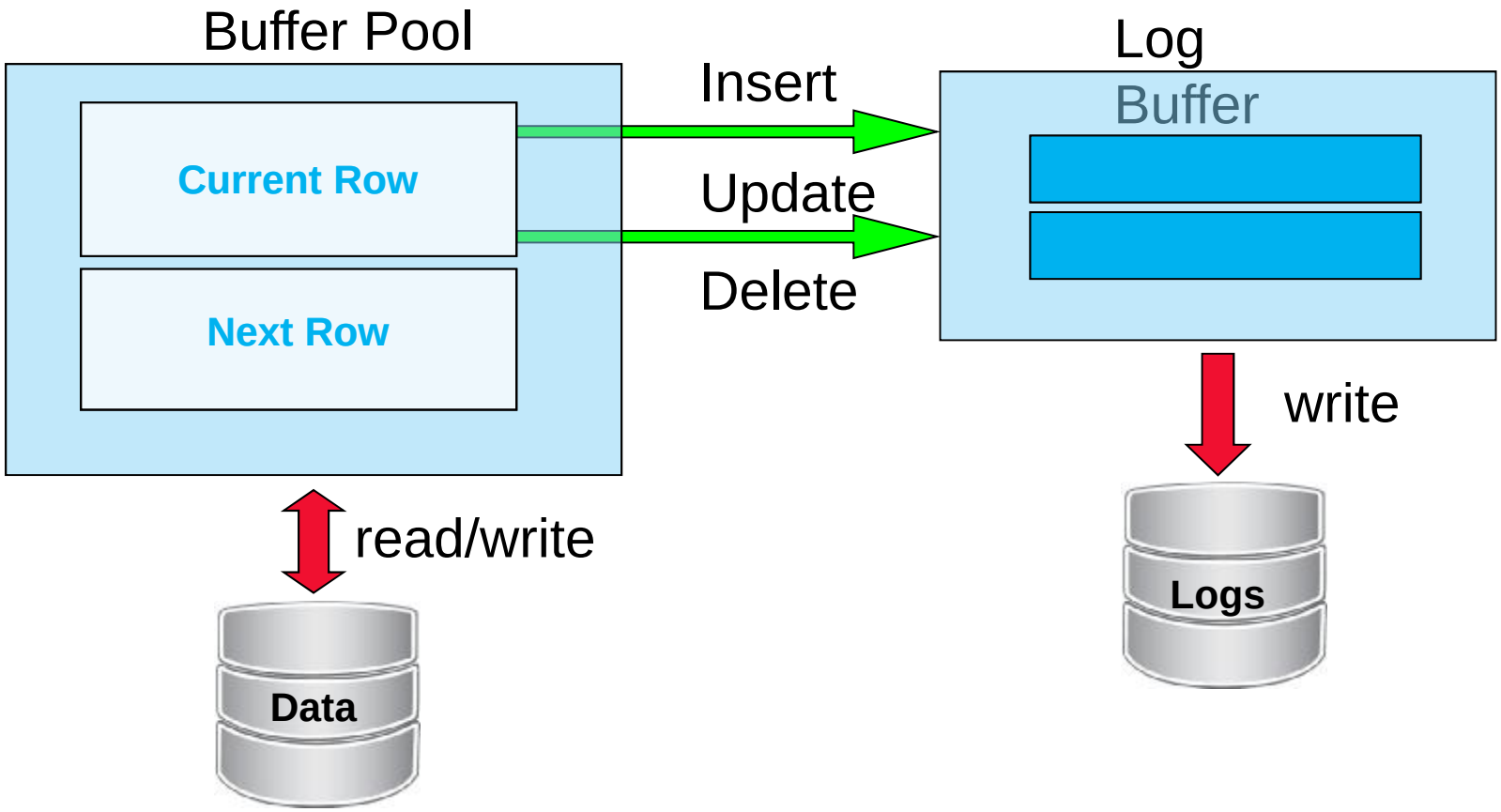
If the Log Buffer is full and

When a commit or rollback is issued.

A change made to a page in the bufferpool cannot be made permanent until the transaction associated with the change has been written to a log file, however a transaction being written to a transaction log file has no direct bearing on when that page is written back out to permanent storage/disk. The timing of writing bufferpool pages to permanent storage is determined by the IO cleaning algorithms, or occurs when a database is deactivated.

Transaction Logging Overview

The process of recording changes to database objects and data



PRESENTATION NOTES FOR PAGE 46

Transaction logging is that process that records each change to a database to allow recovery.

All DB2 databases have logs associated with them. These logs keep a record of all changes made to the database objects and data.

All the changes are first written to log buffers in memory before being externalized to the log files on disk.

All changes to data pages are logged, the update data page is not written to disc before it’s associated log record is written to the log.

Logging performance can have a significant impact on overall database and application performance

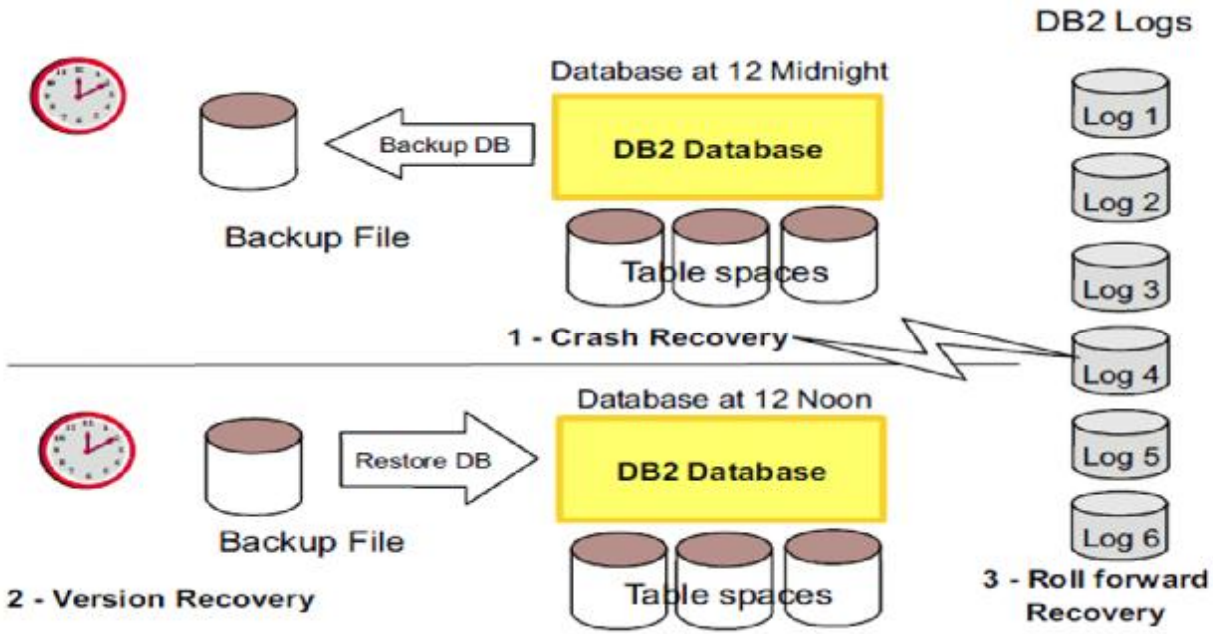
DB2 Logging Concepts

- **Records database transactions**
  - If there is a crash, the Logs are used to undo or redo transactions during recovery
- **Logging is always "ON" for regular tables in DB2**
  - Possible to mark some tables or columns as NOT LOGGED
  - Possible to log or not the USER temporary tables
- **Transaction or Unit of Work (UOW)**
  - A sequence of one or more SQL statements
  - Initiated by the first executable SQL statement after connecting to the database
  - UOW terminates with a COMMIT or ROLLBACK
- **DB2 implements two types of logging**
  - Circular logging (default)
  - Archive logging

PRESENTATION NOTES FOR PAGE 47

DB2 Logging and Recovery Methods

- **Database Crash Recovery (DB2 Logs)**
  - Recovery from unscheduled outages
  - Uses database logs to Undo or Redo changes
- **Version Recovery (DB2 Backup Image)**
  - Recovery of a database to a previous state using a DB2 backup
- **Roll forward Recovery (DB2 Backup Image + DB2 Logs)**
  - Recovery of database or table space changes using a DB2 backup image and then applying the DB2 logs using roll forward



PRESENTATION NOTES FOR PAGE 48

All databases have logs associated with them. These logs keep records of database changes. If a database needs to be restored to a point beyond the last full, offline backup, logs are required to roll the data forward to the point of failure.

Each database includes recovery logs, which are used to recover from application or system failures.

They can be used to apply changes in case of a crash, or to rollback changes during normal operation.

In combination with the database backups, they are used to recover the consistency of the database right up to the point in time when the failure occurred.

A DB2 database always has some kind of logging working on its behalf.

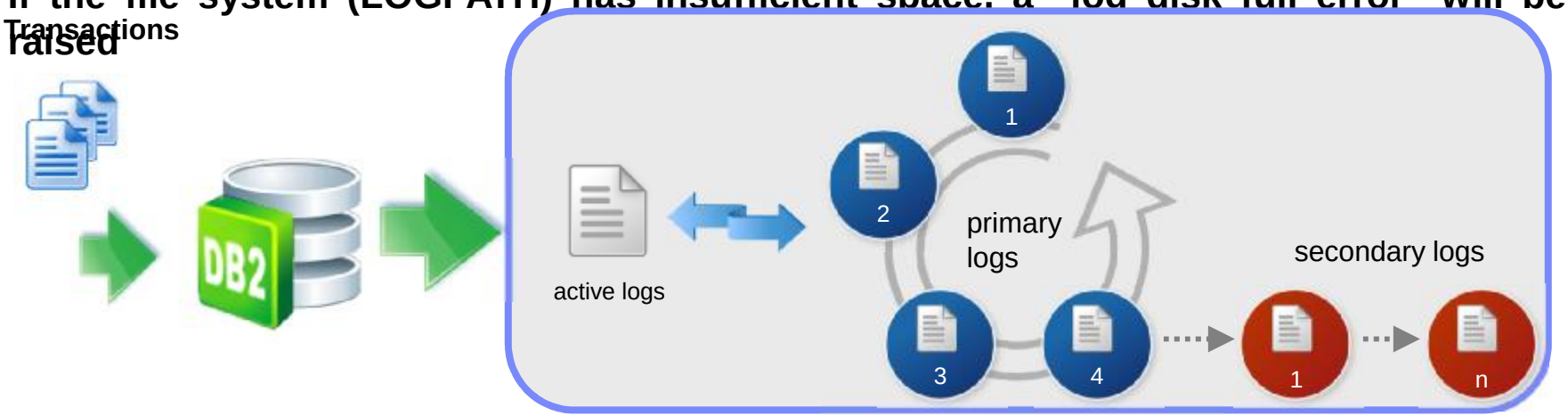
DB2 supports two types of logging: CIRCULAR, the default, and ARCHIVE. We will look at these two types in more detail in the upcoming slides.

All tables being written to are logged, unless a table or column has been defined as NOT LOGGED INITIALLY, or the table is a temporary table. A user defined temporary table can created with the option of being logged or not.

Long Transaction is when the log space is exhausted.

Circular Logging

- **Primary log files (LOGPRIMARY) are used to record all changes and reused when changes are committed**
  - Crash and version recovery possible; roll-forward recovery not possible
  - Only full, offline database backups are allowed
- **Secondary log files (LOGSECOND) allocated when limit of primary log files reached**
  - If both primary and secondary log limit is reached, an error code is returned
- **If the file system (LOGPATH) has insufficient space, a "log disk full error" will be raised**



PRESENTATION NOTES FOR PAGE 49

In the context of performance Circular logging has less impact on performance, primarily due to the fact that there is no archiving of circular logs, log files are reused in a circular fashion.

Circular logging is the default logging behaviour when a new database is created. With this type of logging, logs are retained only to the point of ensuring the integrity of the current transactions. Circular logging only allows for full, offline backups at the database level and during offline backups, the database will be inaccessible to users.

There are two types of log files used, primary and secondary. Primary log files establish a fixed amount of storage allocated to the recovery log files. The DB CFG parameter LOGPRIMARY allows you to specify the number of primary log files to be pre-allocated. LOGSECOND specifies the maximum number of secondary log files that can be created, on an “as needed” basis. The DB CFG parameter LOGPATH specifies the location of the log files.

As the name suggests, circular logging uses a "ring" of logs to provide recovery from transaction failures and system crashes. The primary logs are used repeatedly in sequence; when a log is full, the next primary log in the sequence is used if it is available. A log is considered available if all units of work within it have been committed or rolled-back. If there are no available primary logs, a secondary log is allocated and used. Secondary logs are dynamically de-allocated as they are no longer needed.

With Circular Logging the database’s oldest logs are being overwritten and thus roll-forward recovery can not be accomplished. Only data that is easily recreated or can be restored using version recovery should be stored in a database using this type of logging.

Circular logging used the number of primary log files specified within the database configuration file (LOGPRIMARY). This parameter represents the number of primary log files that will be allocated to support database logging. The log files are used in sequence. A log file can be reused when all units of work contained within it are committed or rolled back.

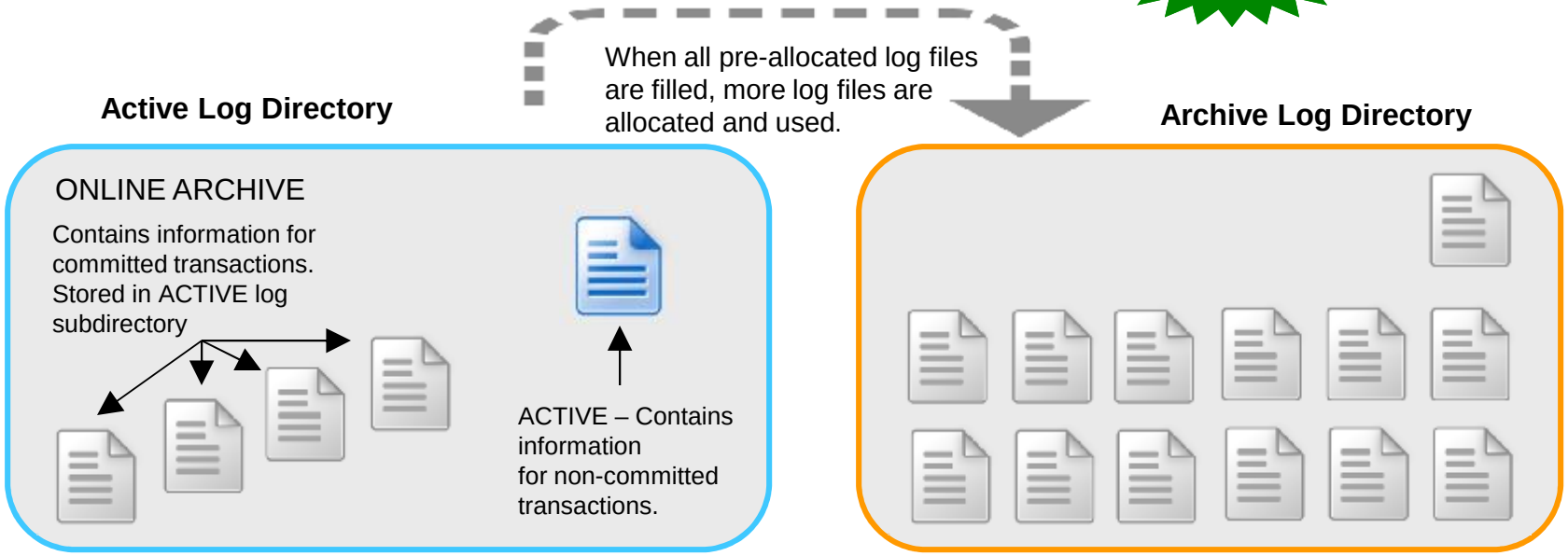
Secondary logs, also specified within the database configuration file (LOGSECOND) are used at a point in time when the database manager requests the next log in the sequence and it is not available for reuse, a secondary log file will be allocated. After it becomes full the next primary log file is checked for reuse again. If not available, another secondary log file is allocated. This process continues until the primary log file becomes available for reuse or the number of secondary log files has been exhausted.

Active logs are used during crash recovery to prevent a failure (system power or application error) from leaving a database in an inconsistent state. Active logs are located in the database log path directory.

Archive Logging

- **Maintain a history of log files**
  - Enable with **LOGARCHMETH1** DB configuration parameter
    - **LOGRETAIN** and **USEREXIT** have been discontinued in DB2 10. They have been replaced with **LOGARCHMETH1**
  - Allows roll-forward recovery or online backup
- **Logs can be archived externally when no longer active to avoid exhaustion of log directory**
- **As of DB2 10, archived log files can be compressed**

NEW IN DB2 10



PRESENTATION NOTES FOR PAGE 50

Archive logging is used specifically for roll-forward recovery or online backups. Archived logs are log files that are copied from the current log path or from the mirror log path to another location.

You can use the logarchmeth1 database configuration parameter, the logarchmeth2 database configuration parameter, or both to enable you or the database manager to manage the log archiving process.

To reduce the amount of disk space required to store archived log files, archived log files can now be compressed in DB2 LUW version 10.

The main benefit of this solution results in reduced storage costs associated with roll-forward recoverable databases. DB2 for Linux, UNIX, and Windows has existing mechanisms to compress data and indexes in the actual database, as well as to compress backup images. This solution adds the capability to compress archived log files. Archived log files are the third major space consumer for roll-forward recoverable databases.

Archived log files contain a considerable amount of data. They can grow quickly, especially for highly concurrent OLTP scenarios. If the modified data is in compressed tables, logging disk space is already reduced by virtue of including compressed record images in the log records. However, there is potential for further storage cost savings if compression is applied to the archived log files themselves.

This feature is available in all DB2 for Linux, UNIX, and Windows editions that support backup compression. Like backup compression, this new function does not require a DB2 Storage Optimization Feature license in DB2 Enterprise Server Edition.

If the logarchmeth1 and logarchmeth2 configuration parameters are set to DISK, TSM, or VENDOR, you can enable archived log file compression by setting the logarchcompr1 and logarchcompr2 configuration parameters to ON. If logarchcompr1 and logarchcompr2 are set dynamically, log files already archived are not compressed.

LOGRETAIN

This parameter has been replaced with logarchmeth1, the primary log archive method configuration parameter. To retain active log files for rollforward recovery, setlogarchmeth1 to LOGRETAIN by issuing the command UPDATE DB CFG USING logarchmeth1 LOGRETAIN. logarchmeth2 must be set to OFF.

USEREXIT

This parameter has been replaced with logarchmeth1, the primary log archive method configuration parameter. To enable log archiving through a user exit program, setlogarchmeth1 to USEREXIT by issuing the command UPDATE DB CFG USING logarchmeth1 USEREXIT. logarchmeth2 must be set to OFF.

Infinite Logging

- Issue with limited number of logs
  - A long running transaction can exhaust logs allocation, even after secondary log files are allocated
  - The number of primary and secondary log files must comply:
    - If logsecond has a value of -1, logprimary **<= 256**
    - If logsecond does not have a value of -1, (logprimary + logsecond) **<= 256**

- Solution: Infinite Logging
- No limit on the size or the number of in-flight transactions running
- Enabled by setting logsecond to -1
- Database must be configured to use archive logging
  - Can hinder performance for rollback and crash recovery
- Other control parameters
    - num_log_span: number of log files an active transaction can span
    - max_log: percentage of the primary log space that a transaction can consume

PRESENTATION NOTES FOR PAGE 51

If you set logsecond to -1, the database is configured with infinite active log space. There is no limit on the size or the number of in-flight transactions running on the database. If you set logsecond to -1, you still use the logprimary and logfilsiz configuration parameters to specify how many log files the database manager should keep in the active log path. If the database manager needs to read log data from a log file, but the file is not in the active log path, the database manager retrieves the log file from the archive to the active log path. (The database manager retrieves the files to the overflow log path, if you have configured one.) Once the log file is retrieved, the database manager will cache this file in the active log path so that other reads of log data from the same file will be fast. The database manager will manage the retrieval, caching, and removal of these log files as required.

You cannot configure infinite active log space in DB2 pureScale environments.

By setting logsecond to -1, you will have no limit on the size of the unit of work or the number of concurrent units of work. However, rollback (both at the savepoint level and at the unit of work level) could be very slow due to the need to retrieve log files from the archive. Crash recovery could also be very slow for the same reason. The database manager writes a message to the administration notification log to warn you that the current set of active units of work has exceeded the primary log files. This is an indication that rollback or crash recovery could be extremely slow.

Rogue queries may generate large logging volume, so that available storage could be consumed very quickly.

Log control files

§ Used to determine which records from the log files need to be applied to the DB when it restarts after a failure

§ Redundancy for database resilience for protection

- Two copies of the each member's log control file, SQLOGCTL.LFH.1 and SQLOGCTL.LFH.2
- Two copies of the global log control file, SQLOGCTL.GLFH.1 and SQLOGCTL.GLFH.2

§ Performance considerations

- Applying the transaction information contained in the log control files contributes to the overhead of restarting a database after a failure
  - Use the **softmax** parameter to configure the frequency at which the database manager writes the buffer pool pages to disk

PRESENTATION NOTES FOR PAGE 52

When a database restarts after a failure, the database manager applies transaction information stored in log files to return the database to a consistent state. To determine which records from the log files need to be applied to the database, the database manager uses information recorded in log control files.

softmax

This parameter determines the frequency of soft checkpoints and the recovery range, which help out in the crash recovery process.

Agenda

- Logging Concepts
- **Configuration and Performance**
- Logging Bottlenecks
- Reducing Logging
- Monitoring and Tuning

PRESENTATION NOTES FOR PAGE 53

Logging Configuration and Performance

LOGARCHMETH1 and LOGARCHMETH2

LOGARCHOPT1 and LOGARCHOPT2

LOGPATH and NEWLOGPATH

MIRRORLOGPATH

OVERFLOWLOGPATH

BLK_LOG_DSK_FUL

MAX_LOG

MINCOMMIT

NUM_LOG_SPAN

FAILARCHPATH

NUMARCHRETRY

ARCHRETRYDELAY

LOGPRIMARY

LOGSECOND

LOGPATH

NEWLOGPATH

LOGPRIMARY

LOGSECONDARY

LOGBUFSZ

LOGFILSIZ

MINCOMMIT

NEW IN DB2 10

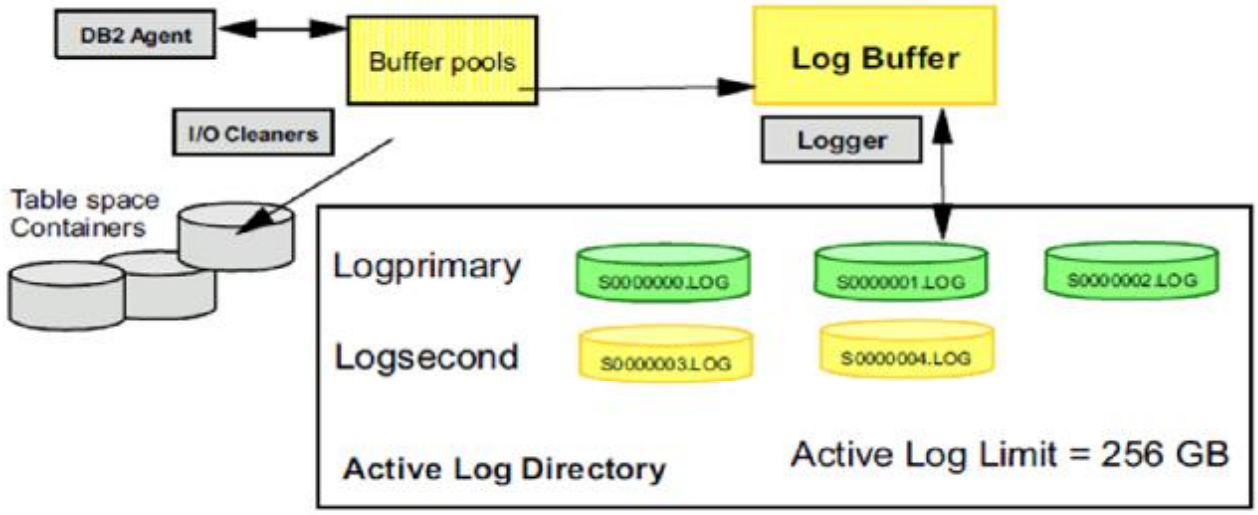
PRESENTATION NOTES FOR PAGE 54

The configuration parameters listed above can impact logging for the database. The next several slides summarize some considerations regarding these parameters. The proper values for these parameters are highly dependent on the installation requirements, so care must be taken regarding rules of thumb.

MINCOMMIT: This parameter is deprecated in Version 10.1 and might be removed in a future release. This parameter can still be used in releases before Version 10.1. In Version 10.1 and later releases, the value specified for this configuration parameter is ignored.

Primary and Secondary Logs

- Primary logs are PREALLOCATED
- Secondary logs are ALLOCATED as needed to handle spikes in workload
- For day to day operations, ensure that you stay within your primary log allocation



	Default	Minimum	Maximum
Logprimary	3	2	256 - Logsecond
Logsecond	2	0	256 - Logprimary

PRESENTATION NOTES FOR PAGE 55

Primary log files are pre-allocated when the database is activated. Deciding on how many primary logs to allocate depends on a number of factors, including the type of logging being used, the size of the log files, length of transactions, and frequency of commits.

A set of one or more log files that are used to record changes to a database and for which storage is allocated in advance.

Secondary log files should be reserved for the periodic need of large amounts of log space. For example, an application that is run once a month might require log space beyond that provided by the primary log files. Since secondary log files do not require permanent file space they are advantageous in this type of situation.

A set of one or more log files used to record changes to a database when the primary log is full.

For daily operations, always try to stay within the limits of your primary logs. If you find that secondary log files are frequently being allocated, you might be able to improve system performance by increasing the log file size or by increasing the number of primary log files.

Primary Logs (LOGPRIMARY)

§ Characteristics

- This parameter specifies the number of primary logs of size *logfilsiz* that will be created
- The primary log files establish a fixed amount of storage allocated to the transaction log files
- A primary log requires the same amount of disk space whether it is full or empty
- The maximum number of primary logs is 256, the default is 3

§ Impact

- One can encounter a “log-full” condition if configured with an insufficient number

PRESENTATION NOTES FOR PAGE 56

LOGPRIMARY Number of primary log files. Default [range] 3 [2 - 256]

This value represents the number of primary log files that will be allocated to support database logging. The appropriate value is highly dependent on installation-specific requirements. For circular logging that is frequently using secondary logs, it may be necessary to increase this value. For log retention logging with highly active systems, it may be necessary to use a value greater than the default so that waits for log allocations are not experienced.

A primary log file, whether empty or full, requires the same amount of disk space. Thus, if you configure more logs than you need, you use disk space unnecessarily. If you configure too few logs, you can encounter a log-full condition. As you select the number of logs to configure, you must consider the size you make each log and whether your application can handle a log-full condition. The total log file size limit on active log space is 256 GB.

If you are enabling an existing database for rollforward recovery, change the number of primary logs to the sum of the number of primary and secondary logs, plus one.

Secondary Logs (LOGSECOND)

§ Characteristics

- If the primary log files become full, secondary log files are allocated, as needed up to the maximum specified by LOGSECOND
- Once allocated, they are not deleted until the database is deactivated
- The maximum number of primary and secondary log files allowed (logprimary + logsecond), gives an upper limit of 1024 GB of active log space
- Setting of -1 for LOGSECOND enables “infinite logging”

§ Impact

- Infinite logging could impact performance if a log file has to be brought back from an archive for ROLLBACK

PRESENTATION NOTES FOR PAGE 57

LOGSECOND Number of secondary log files. Default [range] 2 [-1; 0 – 254]

This parameter specifies the number of secondary log files that are created and used for recovery log files (only as needed). When the primary log files become full, the secondary log files (of size logfilsiz) are allocated one at a time as needed, up to a maximum number as controlled by this parameter. Changes to this parameter are immediately effective.

Setting of -1 for LOGSECOND enables infinite logging. This has to be used with care. If there is a runaway transaction it could exhaust logging space.

With infinite logging there could be a performance hit on roll backs. Log files archived as soon as they are full instead of when they are inactive. Implication is that log file will have to be brought back from archive if transactions is rolled back.

NOTE: The need to avoid allocating a large number of secondary log files, performance is impacted for each allocated log

Log File Size (LOGFILSIZ)

§ Characteristics

- DB CFG parameter defines the size of each primary and secondary log file in 4K pages

§ Impact

- The size of the log file has a direct bearing on performance
  - "Too small"
    - Overhead of archiving more old log files
    - Allocating new log files more frequently
  - "Too big"
    - Logistics of managing large files during archival process
    - Risking the loss of a greater quantity of transactions if a log cannot be recovered

PRESENTATION NOTES FOR PAGE 58

LOGFILSIZ Size of primary and secondary log file in pages. Default [range] UNIX 1000 [4 - 1 048 572]; Intel 1000 [4 - 1 048 572]

This parameter determines the number of pages to be allocated when a log file is requested. Combined with logprimary and logsecond, this value determines the disk space required to support logging. Since all primary logs are allocated, even if not used, the value of this parameter can have major impact on the system disk utilization.

The Log file size (LOGFILSZ) parameter specifies the size of each configured log, in number of 4K pages.

There is a 1024 GB logical limit on the total active log space that you can configure. This limit is the result of the upper limit for each log file, which is 4 GB, and the maximum combined number of primary and secondary log files, which is 256.

The size of the log file has a direct bearing on performance. There is a performance cost for switching from one log to another. So, from a pure performance perspective, the larger the log file size the better. This parameter also indicates the log file size for archiving. In this case, a larger log file is size it not necessarily better, since a larger log file size can increase the chance of failure or cause a delay in log shipping scenarios. When considering active log space, it might be better to have a larger number of smaller log files. For example, if there are 2 very large log files and a transaction starts close to the end of one log file, only half of the log space remains available.

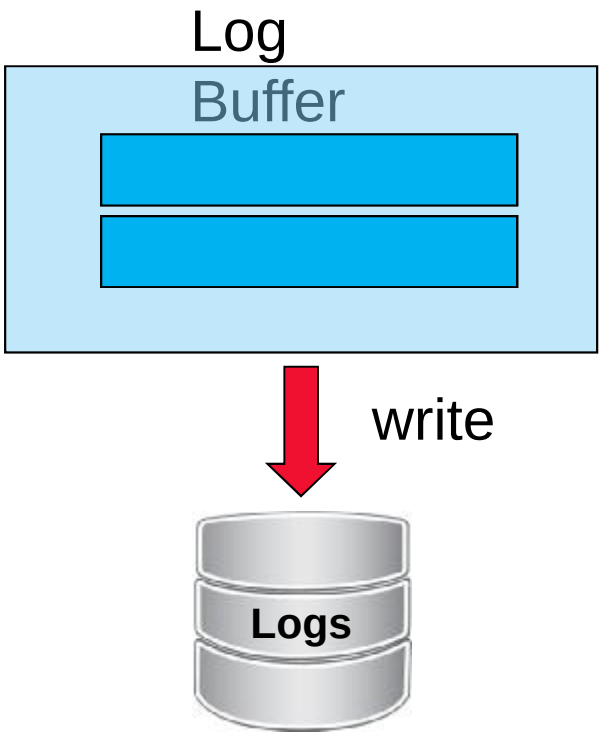
Every time a database is deactivated (all connections to the database are terminated), the log file that is currently being written is truncated. So, if a database is frequently being deactivated, it is better not to choose a large log file size because DB2 will create a large file only to have it truncated. You can use the ACTIVATE DATABASE command to avoid this cost, and having the buffer pool primed will also help with performance.

Assuming that you have an application that keeps the database open to minimize processing time when opening the database, the log file size should be determined by the amount of time it takes to make offline archived log copies.

Minimizing log file loss is also an important consideration when setting the log size. Archiving takes an entire log. If you use a single large log, you increase the time between archiving. If the medium containing the log fails, some transaction information will probably be lost. Decreasing the log size increases the frequency of archiving but can reduce the amount of information loss in case of a media failure since the smaller logs before the one lost can be used.

When Are Log Records Written to Disk?

- **Transaction log records are written from log buffer to log files**
- **Transaction committed**
  - application commits
  - group of transactions commit, as defined by the **mincommit** configuration parameter
- **Log Buffer full**
  - once the internal log buffer becomes full log records are externalized to the log files on disk
- **LSN Gap Trigger**
  - When the amount of log space between the log record that updated the oldest page in the buffer pool and the current log position exceeds that allowed by the softmax database configuration parameter, the database is said to be experiencing an LSN gap
- **SOFTMAX reached**
  - Softmax is a DB CFG parameter which forces a write to disk when exceeded
- **This guarantees recoverability during crash recovery**



MINCOMMIT This parameter is deprecated in Version 10

PRESENTATION NOTES FOR PAGE 59

Transaction committed at the time the application commits or a group of transactions commit, as defined by the mincommit configuration parameter the transaction log records are written from the log buffer to the log files on disk. This guarantees recoverability during crash recovery.

Log Buffer full once the internal log buffer becomes full log records are externalized to the log files on disk.

LSN Gap Trigger an internal database manager event for example when buffer pool I/O page cleaners are activated.

Description: LSN Gap Situation. When the amount of log space used, starting from the time a log record for the oldest dirty page in the buffer pool was written, exceeds the percentage of log file data that can be written before a soft checkpoint will be established, it is said that the database is in an “LSN gap” situation. When the logger detects than an LSN gap has occurred, it will trigger the page cleaners to write out all dirty pages contributing to the LSN gap situation. (That is, it will write out those pages which are older than what is allowed by the softmax database configuration parameter.)

There is another case when Log Records are written to disk - * Write Ahead Logging - Before the corresponding data/index pages are written to disk.

MINCOMMIT: This parameter is deprecated in Version 10.1 and might be removed in a future release. This parameter can still be used in releases before Version 10.1. In Version 10.1 and later releases, the value specified for this configuration parameter is ignored.

Log Buffer Size (LOGBUFSZ)

- DB CFG parameter specifying the amount of memory allocated as a buffer for more efficient log file I/O
  - Not managed by Self-Tuning Memory Manager (STMM)

- Default value usually not large enough for OLTP databases (8 4K pages), 256 (4K pages) is a good starting point:

```
db2 update db cfg for sample using LOGBUFSZ
256
```

PRESENTATION NOTES FOR PAGE 60

LOGBUFSZ Number of pages used to buffer log records. Default 32-bit platforms 8 [4 - 4 096]; 64-bit platforms 8 [4 - 131 070]

This parameter allows you to specify the amount of memory to use as a buffer for log records before writing these records to disk. The log records are written to disk when any one of the following events occurs:

A transaction commits

The log buffer becomes full

Some other internal database manager event occurs.

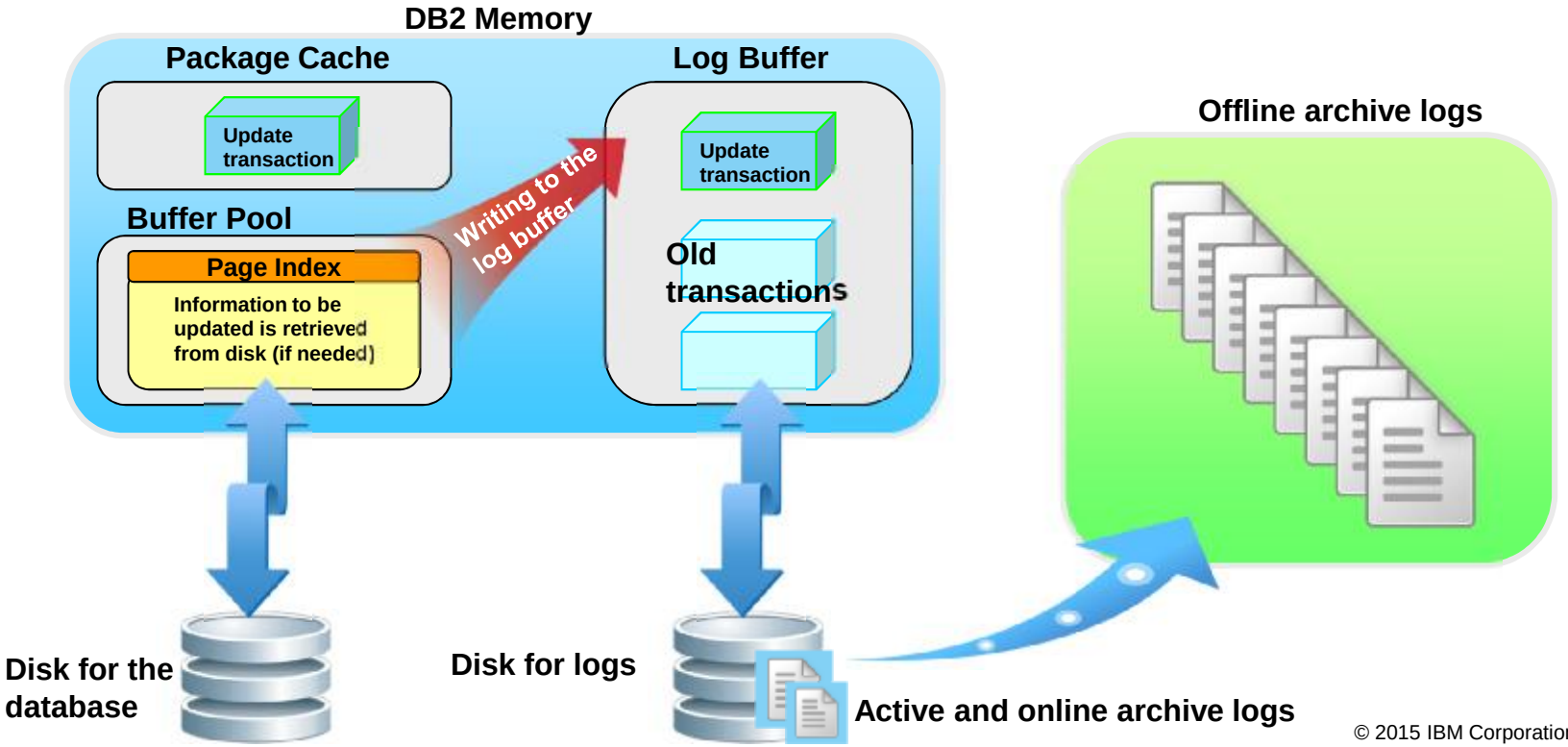
Increasing the log buffer size can result in more efficient input/output (I/O) activity associated with logging, because the log records are written to disk less frequently, and more records are written each time. However, recovery can take longer with a larger log buffer size value. As well, you may be able to use a higher logbufsz setting to reduce number of reads from the log disk. (To determine if your system would benefit from this, use the log_reads monitor element to check if reading from log disk is significant.

Log File States

§ Active logs
  – Contain at least 1 transaction that has not been committed or rolled back

§ Online archive logs
  – Contain committed and externalized transactions in the active log directory

§ Offline archive logs
  – Contain committed and externalized transactions in a separate repository



PRESENTATION NOTES FOR PAGE 61

Regardless of the logging strategy, all changes to regular data and index pages are written to the log buffer. The data in the log buffer is written to disk by the logger process. In the following circumstances, query processing must wait for log data to be written to disk:

On COMMIT

Before the corresponding data pages are written to disk, because DB2(R) uses write-ahead logging. The benefit of write-ahead logging is that when a transaction completes by executing the COMMIT statement, not all of the changed data and index pages need to be written to disk.

Before some changes are made to metadata, most of which result from executing DDL statements

On writing log records into the log buffer, if the log buffer is full

DB2 manages writing log data to disk in this way in order to minimize processing delay. In an environment in which many short concurrent transactions occur, most of the processing delay is caused by COMMIT statements that must wait for log data to be written to disk. As a result, the logger process frequently writes small amounts of log data to disk, with additional delay caused by log I/O overhead.

DB2CKLOG - DB2 CHECK LOG

§ Check the validity of archive log files
- Determine whether the log files can be used during roll-forward recovery
- A single archive log file or a range of archive log files can be checked

```
DB2CKLOG log_num ARCHLOGPATH path
DB2CKLOG log_num to log_num2
```

Validating a range of logs:

```
$ db2cklog 3 to 5
                    ____    D B 2 C K L O G    ____

                        DB2 Check Log File tool

...
"db2cklog": Finished processing log file "S0000003.LOG". Return code: "0".
...
"db2cklog": Finished processing log file "S0000004.LOG". Return code: "0".
...
"db2cklog": Finished processing log file "S0000005.LOG". Return code: "0".
```

Successful Validation

PRESENTATION NOTES FOR PAGE 62

You use the db2cklog command to check the validity of archive log files in order to determine whether or not the log files can be used during rollforward recovery of a database or table space. Either a single archive log file or a range of archive log files can be checked.

Archive log files that pass validation by the db2cklog command without any DBT error messages or warnings can be used during a rollforward recovery operation. If an archive log file fails validation with an error message or if a warning is returned, then you must not use that log file during rollforward recovery. A log file that returns an error during validation by the db2cklog command will cause the recovery operation to fail. If the validation of a log file returns a warning, then that log file might be invalid, unless the log file is still active. Only log files that are closed, such as archive log files, can be validated successfully.

Command parameters:

ARCHLOGPATH path

Specifies a relative or an absolute path where the archive log files are stored. The default path is the current directory.

New Log Path (NEWLOGPATH)

- DB CFG parameter which is used to specify a new location for the log files
  - Set only when relocating the log files; otherwise the parameter has no value
- Ideally, the log files will be on a physical disk not shared with the database or other applications
  - Recommended to use RAID-10 for logs to reduce the chance of losing log files due to disk failures
-

```
db2 update db cfg for sample using NEWLOGPATH /db2logs
```

PRESENTATION NOTES FOR PAGE 63

Separate DB2 transaction logs and data
To help achieve best availability, separate the transaction logs and DB2 data, or table spaces, onto separate spindles and onto separate LUNs.

Each DB2 partition should be provided one LUN with dedicated spindles for its transaction logs and, typically, multiple LUNs for its table space containers or data.

While the ratio of log spindles to data spindles is heavily dependent on workload, a good rule of thumb is 15% to 25% of the spindles for logs, and 75% to 85% of the spindles for data.

Do not use a network or local file system that is shared as the log path for both the primary and standby databases in a DB2 High Availability Disaster Recovery (HADR) database pair

General Recommendations

- Location of Database logs
  - Need to be on their own physical disk
  - A fast I/O device for the log is recommended
  - RAID 10 recommended
- LOGFILSIZ
  - Increase beyond default, e.g. 5000 4K pages or more
- LOGPRIMARY
  - Allocate all logs as primary logs
  - Use LOGSECOND for “emergency” space only
- LOGBUFSZ
  - Increase to 256 or greater
- Use DB2 and operating system tools to monitor logging activity

PRESENTATION NOTES FOR PAGE 64

Location of Database logs - Separate DB2 transaction logs and data to help achieve best availability, separate the transaction logs and DB2 data, or table spaces, onto separate physical disk and onto separate LUNs. Where possible, use RAID-10 for transaction logs. RAID-10 provides excellent redundancy because it can survive multiple physical disk failures. RAID-5, on the other hand, can only survive a single physical disk failure.

LOGFILSIZ – The size of the log file has an impact on performance because there is a cost for switching from one log to another. So, from a pure performance perspective, the larger the logfile size, the better. This parameter also indicate the log file size for archiving. In this case, a larger log file size is not necessarily better, because a larger log file size may increase the chance of failure or cause a delay in log shipping scenarios. When considering the active log space, it may be better to have a larger number of smaller log files.

LOGBUFSZ - Increasing the log buffer size (logbufsiz) and the log file size (logfilsiz) helps insert performance. Buffering the log records results in more efficient log file I/O, because the log records are written to disk less frequently and more log records are written each time. Another important reason for increasing LOGBUFSIZ is minimizing disk reads for currently committed.

MINCOMMIT – This parameter allows you to delay the writing of the log buffer log records to disk until a minimum number of commits have been performed, when the buffer is filled, or at one second intervals. This can reduce the database manager overhead associated with writing log records. Improve performance when you have multiple applications running against a database and many commits are requested by the applications within a very short period of time.

Agenda

- Logging Concepts
- Configuration and Performance
- **Logging Bottlenecks**
- Reducing Logging
- Monitoring and Tuning

PRESENTATION NOTES FOR PAGE 65

Log Bottleneck

- Anything sharing the disks?
- High transaction rate?
- High data volume?
- Too frequent commits?
- Logging too much data?

PRESENTATION NOTES FOR PAGE 66

Can you reduce commit frequency? (db snapshot to verify commits are high; app snapshot to find who is committing so much)

In rare cases, it might be useful to increase MINCOMMIT. This used to have a greater impact in V7, and its impact varies by platform (generally very low on AIX.) The most benefit would be seen in systems where there are hundreds of connections running very small transactions.

Do you need to increase log buffer size? (only relevant if it's quite small; see new snapshot element indicating ‘log buffer full’ condition.)

Logging Bottlenecks – Disk

- Will interfere with all DML activity and cause COMMITs and ROLLBACKs to take longer
- Can be very performance sensitive, especially in an OLTP environment – a good place to use your best hardware
  - Dedicated disks – separate from tablespaces, etc.
  - Fast disks
  - RAID parallelization with small (e.g. 8k) stripe size
  - Fast controller with write caching
- Is anything using the same disks?
  - Can be difficult to determine conclusively
    - Partitioned disks, logical volumes make it difficult to be sure what's on the same disk spindles
  - Check tablespace container paths, database directory, other utilities, etc.

PRESENTATION NOTES FOR PAGE 67

Transaction logs are often the Achilles' heel of a system, because they can often make the difference between a well-performing system and a poorly performing one. This is an important area to configure carefully.

First, never (if you're concerned about performance) let the logs go to their default location, which is under the database directory. It's generally advisable to keep them on their own spindles, and for high-transaction rate systems, use multiple disks in a RAID configuration.

If you suspect a log IO bottleneck (high IO wait time on the log devices, high log IO time in the snapshots), check to make sure nothing else is on the same disks. Start with examine the log path – but you may have to go as far as to get down to the device level. OS tools can help here.

Logging Bottlenecks – High Data Volume

- What is High Data Volume?
  - –iostat (or perfmon) shows larger average I/O
- Possible Remedy
  - –Can you reduce amount of logged data?
    - Alter table design (i.e., group frequently updated columns, ideally at end of the row)
    - Use ALTER TABLE … NOT LOGGED INITIALLY for “bulk” operations
    - Use LOAD utility to insert large amounts of data.
    - Use TRUNCATE command instead of DELETE to empty a table
    - Use Data Compression of data and indexes.
    - Compressed when using compression which can helps reduce I/O traffic
    - If DGTT/CGTTs are being used set NOT LOGGED
    - Larger I/Os can also be due to a poor performing logging disk

PRESENTATION NOTES FOR PAGE 68

Less common is the case where we’re bottlenecked because of high throughput to the log.

Indicated by an average I/O size 'quite a bit larger' than 4k - 8k, 16k, etc.

Can we reduce the amount logged? If we have freedom to change the schema, we might put columns to be updated together at beginning of row
If it’s due to a bulk operation - import, insert w. subselect, etc. – possibly look at NOT LOGGED INITIALLY
If it’s due to LOB operation, possibly look at not logged LOBs

Logging Bottlenecks – High Transaction Rate

§ What is a High Transaction Rate?

- –iostat (or perfmon) shows log device performing greater than 80-100 I/O requests per second and average size ~4 KB

§ Possible Solution

- –Can you reduce commit frequency?
  - • Database snapshot to verify if commits are high
  - • Application snapshot to find out who is committing so frequently
- –Increase log buffer size
  - • May be under-sized
  - • # times log buffer filled, etc.

PRESENTATION NOTES FOR PAGE 69

iostat shows log device performing > 80-100 IOs operations per second and average I/O size ~4 KB on a single disk (higher on RAID stripe sets), small (e.g. 4k) avg size

Can you reduce commit frequency? Identify the number of commits using Application Snapshot, identify who's issuing commits using Database Snapshot

Possibly increase log buffer size? Use SYSIBMADM.SNAPDB NUM_LOG_BUFFER_FULL

Can you increase MINCOMMIT? Default value for MINCOMMIT is 1. Increase if multiple read/write applications typically request concurrent database commits

Agenda

- Logging Concepts
- Configuration and Performance
- Logging Bottlenecks
- **Monitoring and Tuning**
- Reducing Logging

PRESENTATION NOTES FOR PAGE 70

Identifying The Log Files Location

1. Determine the file systems that reside on the system

```
df –k
Filesystem            1K-blocks      Used Available Use% Mounted on
/dev/sda2              10847448   9537288    759132  93% /
/dev                     517576        96    517480   1% /dev
/dev/sdb1              38456308   1837808  34665000   6% /db2fs
```

2. Examine the database configuration parameters ‘Path to log files’

```
db2 get db config for sample | grep -i 'path to log files'
Path to log files = /db2fs/db2inst1/NODE0000/SQL00006/SQLOGDIR/
```

3. Verify that the transaction logs are not sharing filesystems or logical devices.

In this example the transaction logs are sharing the same location as table space containers

```
SELECT SUBSTR(TBSP_NAME,1,20) AS TBSP_NAME, INT(TBSP_ID) AS TBSP_ID,
       SUBSTR(CONTAINER_NAME,1,45) AS CONTAINER_NAME
              FROM SYSIBMADM.CONTAINER_UTILIZATION

TBSP_NAME     TBSP_ID   CONTAINER_NAME
------------  --------  ---------------------------------------------
SYSCATSPACE          0  /db2fs/db2inst1/NODE0000/SAMPLE/T0000000/C000
TEMPSPACE1           1  /db2fs/db2inst1/NODE0000/SAMPLE/T0000001/C000
USERSPACE1           2  /db2fs/db2inst1/NODE0000/SAMPLE/T0000002/C000
```

PRESENTATION NOTES FOR PAGE 71

First determine the file systems that reside on the system, one method would be to perform a Unix “df –k” (disk free) command. Next examine the “Path to log files” setting in the database configuration file. Then verify that the transaction logs are not sharing file systems with other applications or database containers.

SNAPSHOT – Commits and Rollbacks

```
db2 get snapshot for database on sample
```

```
Commit statements attempted                = 11
Rollback statements attempted              = 0
Dynamic statements attempted               = 524
Static statements attempted                = 16
Failed statement operations                = 0
Select SQL statements executed             = 171
Xquery statements executed                 = 0
Update/Insert/Delete statements executed   = 9
DDL statements executed                    = 0
Inactive stmt history memory usage (bytes) = 0
```

How many Commits

How many Dynamic statements

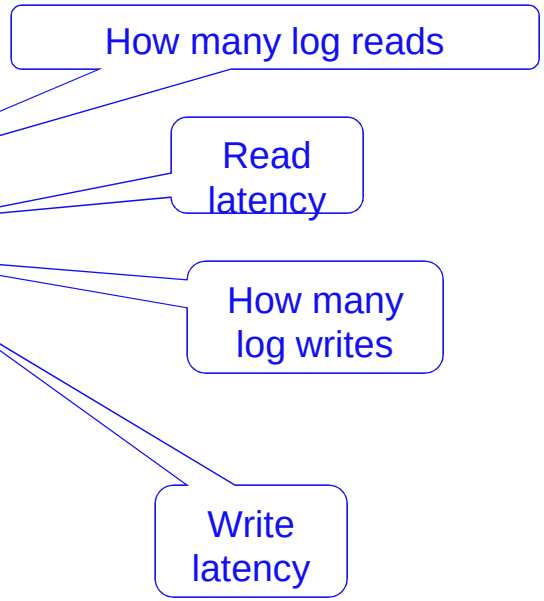
- GET SNAPSHOT FOR database ON sample
- Locate Log section with Commits/Rollback
- Reference Commit, Rollback, Dynamic, Static, etc.
- Trend log information

PRESENTATION NOTES FOR PAGE 72

SNAPSHOT – Log Pages

```
db2 get snapshot for database on sample
```

```
Log space available to the database (Bytes)= 8286039447
Log space used by the database (Bytes)     = 37160553
Maximum secondary log space used (Bytes)   = 0
Maximum total log space used (Bytes)       = 7720996823
Secondary logs allocated currently         = 0
Log pages read                             = 13000
Log read time (sec.ns)                     = 0.000000004
Log pages written                          = 12646941
Log write time (sec.ns)                    = 875.000000004
Number write log IOs                       = 1167739
Number read log IOs                        = 5
Number partial page log IOs                = 105768
Number log buffer full                     = 221
Log data found in buffer                   = 200
```



- **Locate log section**
- **GET SNAPSHOT FOR DATABASE ON ….**
- **Reference log reads and writes**
- **Trend log information:**
  - If there are a large ‘Number Read Log IOs’ relative to ‘Log Data found in buffer’, you need to tune up the LOGBUFSZ
  - If ‘Number of log buffer full’ is high, increase LOGBUFSZ
  - ‘Log write time/’Number write log IOs’ is important. <= 2ms is desirable

PRESENTATION NOTES FOR PAGE 73

Administrative View – LOG_UTILIZATION

```
SELECT substr(db_name, 1,10) DB_NAME,
       log_utilization_percent, total_log_used_kb,
       total_log_available_kb
              FROM SYSIBMADM.LOG_UTILIZATION;
```

```
DB_NAME LOG_UTILIZATION_PERCENT TOTAL_LOG_USED_KB TOTAL_LOG_AVAILABLE_KB
------- ----------------------- ----------------- ----------------------
SAMPLE  21.65                   0                 19902

  1 record(s) selected.
```

Percent utilization of total log space!

This administrative view contains information about log utilization

PRESENTATION NOTES FOR PAGE 74

Administrative View – SNAPDB

```
SELECT VARCHAR(DB_NAME,20) AS DBNAME,
    CASE WHEN (commit_sql_stmts + rollback_sql_stmts) > 0
    THEN DEC((1000 * (log_write_time_s / (commit_sql_stmts +
                      rollback_sql_stmts))),5,0)
    ELSE NULL
    END AS LogWriteTime_PER_1000TRX,
    log_write_time_s AS LOGWTIME,
    commit_sql_stmts + rollback_sql_stmts AS TOTALTRANSACTIONS
 FROM sysibmadm.snapdb;

DBNAME  LOGWRITETIME_PER_1000TRX           LOGWTIME  TOTALTRANSACTIONS
------  ---------------------------------  --------  -----------------
SAMPLE  10                                 20                      2000

  1 record(s) selected.
```

Cumulative average time the agent waited per 1000 transactions

This administrative view contains amount of time an agent waits for log buffer to be flushed

PRESENTATION NOTES FOR PAGE 75

The time an agent waits for log records to be flushed to disk:LOG_WRITE_TIME / TOTAL_COMMITS The transaction log has significant potential to be a system bottleneck, whether due to high levels of activity, or to improper configuration, or other causes. By monitoring log activity, you can detect problems both from the DB2 side (meaning an increase in number of log requests driven by the application) and from the system side (often due to a decrease in log subsystem performance caused by hardware or configuration problems).

Agenda

- Logging Concepts
- Configuration and Performance
- Logging Bottlenecks
- Monitoring and Tuning
- **Reducing Logging**

PRESENTATION NOTES FOR PAGE 76

Reducing the Overhead of Transaction Logging

- NOT LOGGED option for LOB and CLOB data
  - Large object (CLOB) columns are logged by default, if the data they contain is recoverable from outside of the database mark these columns as NOT LOGGED, to reduce the volume of data being logged during insert, update, or delete operations
- ALTER TABLE… NOT LOGGED INITIALLY
  - If the excessive log volume is correlated with bulk SQL operations, the target table can be set to NOT LOGGED INITIALLY
- NOT LOGGED option for temporary tables
  - Declared Global Temporary Tables (DGTTs)
  - Created Global Temporary Tables (CGTTs)
- Use LOAD utility for inserting large amounts of data
  - Load does not go through the SQL engine, therefore it is high speed and logging can be avoided

PRESENTATION NOTES FOR PAGE 77

Reducing the Overhead (continued…)

- Reduce the number of COMMITs
  - Modify the applications such that commits are performed less often
- Grouping frequently updated columns
  - Placing columns that are frequently modified next to one another in the table definition
  - This can reduce the volume of data that is logged during updates
  - They are ideally defined at the end of the row's definition
- Use TRUNCATE to empty a table
  - Truncating a table will avoid the logging activity of DELETE
- Use Data Compression to compress data and indexes
  - Log records are also compressed when using compression which reduces I/O traffic

PRESENTATION NOTES FOR PAGE 78

Database logging

- Rules of thumb
  - Use archive logging in production environments to be able to perform many recovery operations including, online backup, incremental backup, online restore, point-in-time rollforward, and issuing the **RECOVER DATABASE** command
  - Configure secondary log files to provide additional log space on a temporary basis
  - Compress archive logs
  - Consider the I/O adapter or bus bandwidth requirements for transaction logging

PRESENTATION NOTES FOR PAGE 79

Summary

§ In this Module you learned about:

- Locking and concurrency issues can have a significant impact on the performance of a DB2 application
- It is necessary to collect information that would help to identify what type of lock event is involved
- Commit DML (insert, update, delete) and DDL actions as soon as possible
- Avoid concurrent DDL operations if possible
- Avoid using higher isolation levels than necessary
- Use DB2 options for monitoring locking
- DB2 logging concepts
- DB2 logging configuration and performance tuning

PRESENTATION NOTES FOR PAGE 80

The next steps…

PRESENTATION NOTES FOR PAGE 81

The Next Steps…

- Complete the Hands on Lab for this module
  - Log onto SKI, go to “My Learning” page, and select the “In Progress” tab.
  - Find the module
  - Download the workbook 10304_WB1_DB2_Locking_for_Performance.pdf, and 10304_WB2_DB2_Logging_for_Performance.pdf and the virtual machine image 10300_VM1_DB2_PerformanceMonitoringAndTuning_10.5.part#.rar.
  - Follow the instructions in the workbook to complete the lab
- Complete the online quiz for this module
  - Log onto SKI, go to “My Learning” page, and select the “In Progress” tab.
  - Find the module and select the quiz
- Provide feedback on the module
  - Log onto SKI, go to “My Learning” page
  - Find the module and select the “Leave Feedback” button to leave your comments

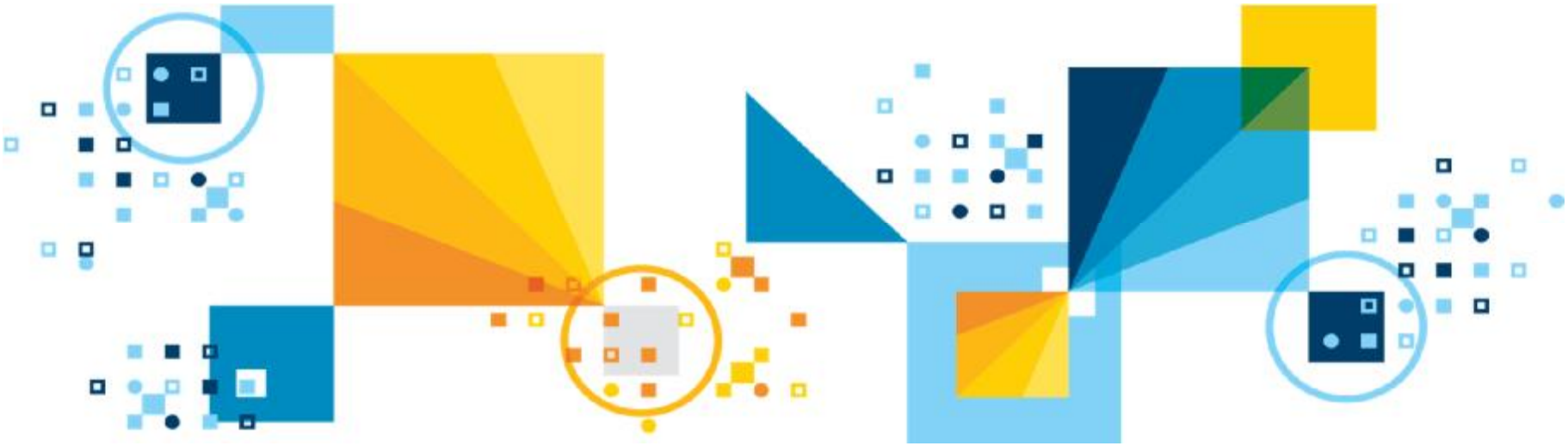
PRESENTATION NOTES FOR PAGE 82

The Next Steps…

- The next set of modules to consider :
  - Module DB2 Maintenance Utilities and Performance

PRESENTATION NOTES FOR PAGE 83

Questions?
askdata@ca.ibm.com



PRESENTATION NOTES FOR PAGE 84