Db2 13 performance and instrumentation

Internal block fetch
AI in Db2 insert/update/delete
AI in Db2 sort optimization
AI in Db2 SORTL
FTB key sizes
Longest lock/latch waiter info
IFCID 306
More latch levels
RTS BIGINT and LOCKMAX

Internal block fetch enhancement

FL 100

Historically, Db2 data manager (DM – aka stage 1) would pass a query's result set one row at a time to the Db2 relational data system (RDS – aka stage 2)

Db2 12 performance enhancement – *internal block fetch*: for certain types of query, DM can pass a set of result set rows to a buffer for retrieval by RDS

With Db2 12, buffer used to hold a block of rows will not be allocated until number of rows fetched from result set has exceeded an internal threshold

- This has the effect of delaying efficiency gain provided by internal block fetch

Db2 13 enhancement: "intelligent" internal block fetch

- Db2 records number of rows fetched for a query, and uses that information to learn that a given query is likely to fetch enough rows for internal block fetch to be helpful
- When that query is subsequently executed, internal block fetch will be immediately enabled
- Use of internal block fetch "from the start" improves efficiency of query execution

AI in Db2: Insert/update/delete performance (index)

FL 100

Previous behavior:

Decision on use of index look-aside based on catalog statistics, which may be stale

Db2 13 behavior:

- Use of index look-aside based on execution-time information (cache of previous index probe)
- Can be used with or without FTBs
- 5-25% CPU savings for batch processing
- 2-3% CPU savings for OLTP

AI in Db2: Sort optimization

FL 100

Previous behavior:

- If any column is a DECFLOAT column, sort does not generate machine code to help move data into sort tree
- Sort enhancements for ORDER BY and GROUP BY are not applied to other areas of sort

Db2 13 behavior:

Improved query performance

- Machine language code generated to boost efficiency of DECFLOAT sorts
- Performance enhancements for DISTINCT, GROUPING SET and PERCENTILE sorts
- More space-efficient sorting for LISTAGG within SUBSTR

Add support to reduce the length of long VARCHAR values if at the end of keys to be sorted (ML enabled)

AI in Db2: SORTL expansion using learning from execution

FL 100

Previous behavior:

- Db2 12 APAR PH31684 added support for SORTL instruction introduced with z15, but Db2 12 use of SORTL was limited
  - Length of sort key could not exceed 136 bytes
  - Length of records to be sorted could not exceed 256 bytes

Db2 13 behavior:

- Db2 13 goes beyond Db2 12 limits, makes SORTL usable in more situations
- Expanded use of SORTL driven by learned behavior of queries:
  - number of rows processed
  - size of row
  - sort pool size

Fast traverse block (FTB)

FL 500

Previous behavior

- Db2 12 introduced in-memory feature: fast index traversal (known as fast traverse block, or FTB)
  - Unique indexes limited to key size of 64 bytes
  - Non-unique indexes limited to key size of 56 bytes

Db2 13 behavior

- New eligibility in key sizes
  - Unique indexes key size limit increased to **128** bytes
  - Non-unique indexes key size limit increased to **120** bytes

Longest lock/latch waiter information

FL 100

Previous behavior:

- A process can wait a long time for a Db2 lock or latch, seriously impacting performance
- Lock and latch information provided via IFCID 148, but that information is available only through a synchronous request from a monitor
- “Standard” Db2 accounting trace information provided average lock/latch wait times for processes, but provided no information about longest latch or lock wait time
  - Result: hard to pinpoint problems in this area

Db213 behavior:

- New “longest lock/latch waiter” information in IFCID 3 (accounting trace record – typically active at all times)
  - Longest page latch wait for a transaction or job (also longest wait for sync and async I/O, and service task)
- This new information can help to identify performance tuning opportunities

Log reading from table space partition level (1|3)

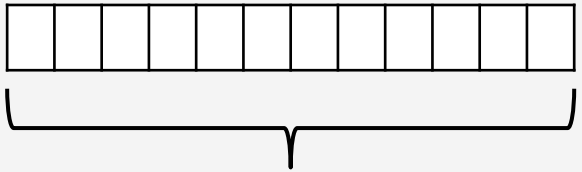
FL 100

For (IIDR) Q Replication,

Q Replication from partitioned table spaces can now include the following scenarios:

1. All partitions replicated
2. Selected partitions replicated (individual and/or ranges)
3. Multiple Q subscriptions and/or queue maps against different partitions and/or partition ranges to improve parallelism, throughput, and latency

Example: Partitioned table space needs parts 1 to 4 to be replicated.



All parts get read for replication

- All logs across the partitioned table space are sent for each Q subscription.
- This may add unnecessary processing:
  - Increases CPU (Db2 and Q Capture)
  - Increases latency
  - Reduces throughput for scenarios #2 and #3

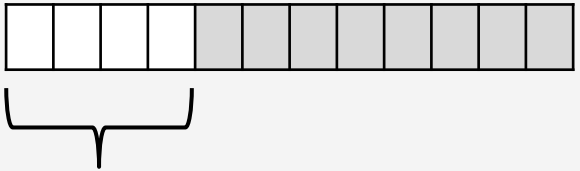
Log reading from table space partition level (2|3)

FL 100

Db2 13 behavior:

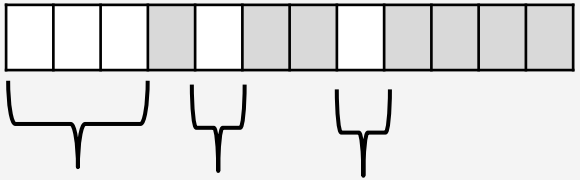
- New log read function uses partition level filtering
- Used only if the table space is partitioned
- Multiple Q Replication processes run more efficiently in parallel to read and process each partition
  - Improves performance for log read throughput
    - Less log data processed by Db2 and Q Capture
  - Reduces CPU requirements
  - Improves latency
  - Increases throughput rate

Example: Partitioned table space needs parts 1 to 4 to be replicated



Parts 1-4 get read for replication

Example: Partitioned table space needs parts 1-3, 5, 8 to be replicated



Those parts get read for replication

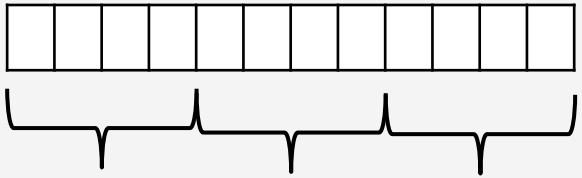
Log reading from table space partition level (3|3)

FL 100

Db2 13 behavior:

- New log read function uses partition level filtering
- Used only if the table space is partitioned
- Multiple Q Replication processes run more efficiently in parallel to read and process each partition
  - Improves performance for log read throughput
    - Less log data processed by Db2 and Q Capture
  - Reduces CPU requirements
  - Improves latency
  - Increases throughput rate

Example: Partitioned table space needs all parts to be replicated.



Parts 1-4 get read in parallel with parts 5-8 and parts 9-12

New support in IFCID 306: EDITPROC

FL 100

Previous behavior:

- Replicating Db2 for z/OS data could require decoding data encoded by an EDITPROC
- IFCID 306 (used for log reads to capture data changes for replication purposes) provided EDITPROC decode functionality, but only for IBM InfoSphere Data Replication (IIDR)
- If another vendor wanted access to that IFCID 306 functionality for their replication tool, vendor had to submit request to IBM Db2 for z/OS development
  - Implemented this way so as not to require change in replication tools that had other EDITPROC decoding mechanisms

Db2 13 behavior:

- No request to IBM Db2 for z/OS development needed in order to utilize EDITPROC decode functionality of IFCID 306
  - Any user of IFCID 306 (in non-proxy mode) can turn on (or turn off) EDITPROC decode functionality

Preparation for future enhancements

FL 100

Increase sizes of various control blocks related to LOBs and the Db2 Data Manager, to support future development and serviceability

- No external changes required

Increase the number of latch levels

- Latching is a Db2 serialization mechanism
  - Multiple latch classes can map to the same latch level, leading to contention and complicating problem diagnosis
- Db2 13 increases number of latch levels from 32 to 64
  - Will enable reduced latch contention, better contention diagnosis
- No external changes required

Real time statistics (RTS) catalog change

FL 500

Current behavior

- Many columns in RTS defined as INTEGER (INT)
  - Max value 2,147,483,647
    - Not large enough in some cases
- Maximum locks on related table spaces based on NUMLKTS
  - Can lead to lock escalation problems:
    - Concurrency
    - Performance

Db2 13 behavior

- Most INTEGER columns changed to BIGINT
  - BIGINT = ± 9 quintillion (9,223,372,036,854,775,807)
  - DSNACCOX supports new value ranges
- LOCKMAX for RTS table spaces change
  - from -1 (LOCKMAX SYSTEM)
  - to 0 (no lock escalation)
- Requires function level V13R1M500 with catalog level at V13R1M501
  - CATMAINT UPDATE LEVEL V13R1M501

RTS tables and table space changes

FL 500

RTS tables (BIGINT changes)

- SYSTABLESPACESTATS
- SYSTABSPACESTATS_H
- SYSINDEXSPACESTATS
- SYSIXSPACESTATS_H

RTS table spaces (LOCKMAX changes)

- SYSTSTSS
- SYSTSTSH
- SYSTSISS
- SYSTSISH

- EXTENTS column from SMALLINT to INTEGER

- Check tooling that relies on RTS for compatibility