Provide a better way for chunking a dump of tables with composite primary key in MySQL Shell

Status: Draft
Roadmap Section: Other
Primary Contact: Kamil Hołubicki
Email: kamil.holubicki@percona.com
Company / Organization: Percona
Role / Team: Staff Software Engineer
Additional Authors / Contributors: None listed
Date: 2026-05-08
Target Release: TBD
Related References: https://bugs.mysql.com/bug.php?id=119222

---

1. High-Level Description

Executive Summary

This is the improvement for MySQL Shell.

MySQL Shell provides util.dumpInstance() utility for creating a logical dump. Instead of keeping everything in a large file, the dump can be chunked into multiple files by multiple threads in parallel. With 'bytesPerChunk' parameter it is possible to specify the required size of dump files chunks.
If the table contains an INT-like primary key, dumpInstance() uses the optimized algorithm, which avoids a full table scan for determining chunks boundaries. It considers only the 1st index column. Such an approach can produce very size-unbalanced chunks for composite primary key and certain data distributions, drastically affecting dump performance. In particular, if one chunk is huge comparing to the others, the dump process becomes single-threaded very soon.
The scope of this work is to extend the current functionality so that other columns of INT-like composite primary keys are used for chunking.

User / Developer Stories

For composite primary keys on columns of integer type, the user should be able to specify the maximum allowed primary key prefix length used for data chunking. util.dumpInstance() command should make the best effort to create chunks of size approximately equal to bytesPerChunk parameter value.

Scope

Extend the current algorithm to allow chunking using multiple columns in the composite key.

Out of Scope / Limitations

Improvement is related only to integer-type columns.

References

https://bugs.mysql.com/bug.php?id=119222

The 1st working implementation of the idea is available on the branch https://github.com/kamil-holubicki/mysql-shell/tree/PS-10413-clean . It is based on 9.7 tag.

---

2. Requirements

Functional Requirements

FR1: Composite INT-like Primary Key Chunking Support

util.dumpInstance() shall support chunking based on composite primary keys where all participating key columns are INT-like types. Supported INT-like types should include:
- TINYINT
- SMALLINT
- MEDIUMINT
- INT
- BIGINT

FR2: Configurable Maximum Primary Key Prefix Length

A new dump option shall allow the user to specify the maximum number of composite primary key columns that may be used for chunk boundary calculation.

FR3: Best-Effort Balanced Chunk Generation

The chunking algorithm shall make a best effort to produce chunks whose estimated data size is approximately equal to the configured bytesPerChunk value.
The implementation is not required to guarantee exact equality.

FR4: Prefix Length Limitation

The algorithm shall never use more columns than:
- the configured maximum prefix length, or
- the actual number of columns in the primary key.

FR5: Non-Eligible Tables Fallback

If a table does not meet eligibility requirements for optimized chunking, the utility shall fall back to the existing generic chunking mechanism.

FR6: Deterministic Chunk Ordering

Chunks generated using composite primary keys shall:
- preserve deterministic ordering,
- avoid overlaps,
- avoid gaps,
- guarantee complete row coverage.

FR7: Statistics-Based Estimation

The chunking mechanism should use available index statistics and metadata to estimate chunk boundaries without requiring a full table scan whenever possible.

FR8: Graceful Degradation for Skewed Data

For heavily skewed key distributions, the implementation shall:
- continue generating valid chunks,
- attempt to minimize chunk size imbalance

FR9: User Visibility

The dump process shall expose chunking decisions in verbose/debug logging.

FR10: Validation of User Input

Invalid prefix-length values shall produce a clear validation error.

FR11: Compatibility with Existing Dump Features

Composite-key chunking shall remain compatible with:
- parallel dumping,
- compression,
- cloud/object storage targets,
- consistency options,

- filtering options,
- table selection options.

Non-Functional Requirements

NFR1: Performance Improvement

For eligible composite INT-like primary keys, the enhanced algorithm should significantly reduce chunk size imbalance compared to the current first-column-only approach.
By doing this, it should also provide full utilization of all dump threads during the time of dump (no single thread dumping the huge chunk while other threads already finished)

NFR2: Avoidance of Full Table Scans

The implementation should avoid full table scans for chunk boundary estimation whenever index-based approaches are feasible.

NFR3: Scalability

The solution shall scale to:
- very large tables,
- billions of rows,
- high-cardinality composite keys.

NFR4: Minimal Additional Memory Usage

The implementation shall not require loading substantial portions of table data into memory.

NFR5: Low Metadata Query Overhead

Additional metadata/statistics queries introduced by the enhancement should have minimal impact on server load.

NFR6: Backward Compatibility

Existing scripts and automation using util.dumpInstance() shall continue functioning without modification unless the new feature is explicitly enabled.

NFR7: Reliability

The implementation shall guarantee:
- no row duplication,
- no row omission,
- deterministic chunk coverage.

NFR8: Concurrency Safety

The enhancement shall behave correctly under existing dump consistency guarantees and transaction isolation mechanisms already supported by MySQL Shell.

NFR9: Observability

Diagnostic logging should provide sufficient information for:
- troubleshooting imbalance,
- analyzing chunk generation,
- understanding fallback behavior.

NFR10: Maintainability

The implementation should:
- isolate chunking logic cleanly,
- minimize duplication with existing algorithms,
- allow future extension to non-integer composite keys if desired.

NFR15 — Failure Transparency

If optimized composite-key chunking cannot be applied, the system shall:

- clearly indicate the reason,
- document the fallback behavior,
- continue dump execution whenever possible.

---

3. Impact Checklist

- ☑ Configuration options or system variables
- ☑ Command-line options or utilities
- ☑ User-visible behavior
- ☑ Performance or resource usage

---

4. High-Level Specification

Summary of the Approach

Existing part of the algorithm will be extended.

User Interface

Configuration / Knobs — New configuration clauses or options

None

Configuration / Knobs — New system variables or command-line options

None

Configuration / Knobs — New command-line options for utilities

NAME: maxKeyPrefixLength
VALUES: integer value
DEFAULT: 1 - means: use only one column
DESCRIPTION: The maximum number of key columns from the left of the composite primary key that may be used for chunking. 0 means use the whole length of the key. Default 1 is to keep legacy behavior by default

NAME: adaptiveStepStrategy
VALUES: original, enhanced
DEFAULT: original
DESCRIPTION: This parameter exists to preserve the legacy algorithm for calculating the adaptive step for chunking. The updated implementation replaces it with a simpler, principle-based heuristic method that provides correct and consistent results.

Configuration / Knobs — New UDFs or similar extension points

None

New Statements

None

Observability

When the dump starts, the log message is displayed informing about maxKeyPrefixLength and adaptiveStepStrategy used for chunking.

User Procedure

The existing parameters list will be extended, like:

util.dumpInstance(“./dumptest.test”, { threads: 32, dryRun: false, consistent: true, compatibility: [“strip_definers”], bytesPerChunk: “1000M”, ocimds: false, showProgress: true, ***maxKeyPrefixLength: 0, adaptiveStepStrategy: “enhanced”***});

Security Context

Nothing beyond existing one.

Compatibility and Behavior Change

Not specifying new parameters causes the tool to fallback to the default/legacy behavior.

---

5. Low-Level Design

Block Diagram

TBD

Interface Specification

See new parameters specification

Design / Implementation Steps

Chunking Algorithm Overview

The algorithm extends the original one that is based on EXPLAIN SELECT statement.

The chunking mechanism divides large table into manageable chunks to enable parallel data extraction and optimal memory usage.
The algorithm supports two strategies: ORIGINAL and ENHANCED. ORIGINAL is the default/legacy behavior of mysqlsh.

Integer Column Chunking:

Phase 1: Range Expansion (Linear)
- Starts with an estimated step size based on: index_range / estimated_chunks
- Expands the search range until enough rows are found
(rows >= rows_per_chunk) - Stops if maximum range is reached or sufficient rows are found

Phase 2: Binary Search (Shrinking)
- If too many rows are found (rows > rows_per_chunk + accuracy)
- Uses binary search to narrow the range to match target row count
- Continues until: delta <= accuracy OR range shrinks to 1
- Falls back to nested chunking if single value still exceeds row limit

Nested Chunking (Deep Chunking):
When a single value in the current key part exceeds rows_per_chunk:
- Recursively chunks the next key part with boundary condition
- Only applies if next column is optimizable (INT-like type)

Chunk Gluing (ENHANCED strategy only):
The Gluer class merges small chunks to optimize dump file count:
- Accumulates consecutive chunks when row count < max_rows_cnt
- Flushes when accumulated size > 3 * max_rows_cnt or at table end
- Prevents fragmentation by combining undersized chunks
- DummyGluer (for ORIGINAL) disables this optimization

---

6. QA Notes

The basic test scenario is described in https://bugs.mysql.com/bug.php?id=119222
QA should test it using huge tables with composite primary keys and uneven data distribution.
Note that it may be the 1st, last or in-the-middle 1st-column that has huge data amount referenced by it.
There may be distributions like: small - huge - small - huge and other various combinations.