is exceeded, an undo tablespace can still be made inactive, but it is not truncated until after the next checkpoint.

`INNODB_METRICS` counters associated with defunct undo truncate flushing operations were removed. Removed counters include: `undo_truncate_sweep_count`, `undo_truncate_sweep_usec`, `undo_truncate_flush_count`, and `undo_truncate_flush_usec`.

See Section 15.6.3.4, “Undo Tablespaces”.

- As of MySQL 8.0.22, the new `innodb_extend_and_initialize` variable permits configuring how `InnoDB` allocates space to file-per-table and general tablespaces on Linux. By default, when an operation requires additional space in a tablespace, `InnoDB` allocates pages to the tablespace and physically writes NULLs to those pages. This behavior affects performance if new pages are allocated frequently. You can disable `innodb_extend_and_initialize` on Linux systems to avoid physically writing NULLs to newly allocated tablespace pages. When `innodb_extend_and_initialize` is disabled, space is allocated using `posix_fallocate()` calls, which reserve space without physically writing NULLs.

  A `posix_fallocate()` operation is not atomic, which makes it possible for a failure to occur between allocating space to a tablespace file and updating the file metadata. Such a failure can leave newly allocated pages in an uninitialized state, resulting in a failure when `InnoDB` attempts to access those pages. To prevent this scenario, `InnoDB` writes a redo log record before allocating a new tablespace page. If a page allocation operation is interrupted, the operation is replayed from the redo log record during recovery.

- **Character set support.** The default character set has changed from `latin1` to `utf8mb4`. The `utf8mb4` character set has several new collations, including `utf8mb4_ja_0900_as_cs`, the first Japanese language-specific collation available for Unicode in MySQL. For more information, see Section 10.10.1, “Unicode Character Sets”.

- **JSON enhancements.** The following enhancements or additions were made to MySQL's JSON functionality:

  - Added the `->>` (inline path) operator, which is equivalent to calling `JSON_UNQUOTE()` on the result of `JSON_EXTRACT()`.

    This is a refinement of the column path operator `->` introduced in MySQL 5.7; `col->>"$.path"` is equivalent to `JSON_UNQUOTE(col->"$.path")`. The inline path operator can be used wherever you can use `JSON_UNQUOTE(JSON_EXTRACT())`, such `SELECT` column lists, `WHERE` and `HAVING` clauses, and `ORDER BY` and `GROUP BY` clauses. For more information, see the description of the operator, as well as JSON Path Syntax.

  - Added two JSON aggregation functions `JSON_ARRAYAGG()` and `JSON_OBJECTAGG()`. `JSON_ARRAYAGG()` takes a column or expression as its argument, and aggregates the result as a single `JSON` array. The expression can evaluate to any MySQL data type; this does not have to be a `JSON` value. `JSON_OBJECTAGG()` takes two columns or expressions which it interprets as a key and a value; it returns the result as a single `JSON` object. For more information and examples, see Section 12.20, “Aggregate Functions”.

  - Added the JSON utility function `JSON_PRETTY()`, which outputs an existing `JSON` value in an easy-to-read format; each JSON object member or array value is printed on a separate line, and a child object or array is intended 2 spaces with respect to its parent.

    This function also works with a string that can be parsed as a JSON value.

    For more detailed information and examples, see Section 12.18.8, “JSON Utility Functions”.