the *N*th expression in the select list; the results are no longer ordered, as is expected with `ORDER BY constant`.

Preparing a statement used as a prepared statement or within a stored procedure only once enhances the performance of the statement, since it negates the added cost of repeated preparation. Doing so also avoids possible multiple rollbacks of preparation structures, which has been the source of numerous issues in MySQL.

For more information, see Section 13.5.1, “PREPARE Statement”.

- **RIGHT JOIN as LEFT JOIN handling.** As of MySQL 8.0.22, the server handles all instances of `RIGHT JOIN` internally as `LEFT JOIN`, eliminating a number of special cases in which a complete conversion was not performed at parse time.

- **Derived condition pushdown optimization.** MySQL 8.0.22 (and later) implements derived condition pushdown for queries having materialized derived tables. For a query such as `SELECT * FROM (SELECT i, j FROM t1) AS dt WHERE i > constant`, it is now possible in many cases to push the the outer `WHERE` condition down to the derived table, in this case resulting in `SELECT * FROM (SELECT i, j FROM t1 WHERE i > constant) AS dt`.

  Previously, if the derived table was materialized and not merged, MySQL materialized the entire table, then qualified the rows with the `WHERE` condition. Moving the `WHERE` condition into the subquery using the derived condition pushdown optimization can often reduce the number of rows must be be processed, which can decrease the time needed to execute the query.

  An outer `WHERE` condition can be pushed down directly to a materialized derived table when the derived table does not use any aggregate or window functions. When the derived table has a `GROUP BY` and does not use any window functions, the outer `WHERE` condition can be pushed down to the derived table as a `HAVING` condition. The `WHERE` condition can also be pushed down when the derived table uses a window function and the outer `WHERE` references columns used in the window function's `PARTITION` clause.

  Derived condition pushdown is enabled by default, as indicated by the `optimizer_switch` system variable's `derived_condition_pushdown` flag. The flag, added in MySQL 8.0.22, is set to `on` by default; to disable the optimization for a specific query, you can use the `NO_DERIVED_CONDITION_PUSHDOWN` optimizer hint (also added in MySQL 8.0.22). If the optimization is disabled due to `derived_condition_pushdown` being set to `off`, you can enable it for a given query using `DERIVED_CONDITION_PUSHDOWN`.

  The derived condition pushdown optimization cannot be employed for a derived table that contains a `UNION` or `LIMIT` clause. In addition, a condition that itself uses a subquery cannot be pushed down, and a `WHERE` condition cannot be pushed down to a derived table that is also an inner table of an outer join. For additional information and examples, see Section 8.2.2.5, “Derived Condition Pushdown Optimization”.

- **Non-locking reads on MySQL grant tables.** As of MySQL 8.0.22, to permit concurrent DML and DDL operations on MySQL grant tables, read operations that previously acquired row locks on MySQL grant tables are executed as non-locking reads.

  The operations that are now performed as non-locking reads on MySQL grant tables include:

  - `SELECT` statements and other read-only statements that read data from grant tables through join lists and subqueries, including `SELECT ... FOR SHARE` statements, using any transaction isolation level.

  - DML operations that read data from grant tables (through join lists or subqueries) but do not modify them, using any transaction isolation level.