SELECT

Syntax

```
SELECT
    [ALL | DISTINCT | DISTINCTROW]
    [HIGH_PRIORITY]
    [STRAIGHT_JOIN]
    [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
    [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [ FROM table_references
      [WHERE where_condition]
      [GROUP BY {col_name | expr | position} [ASC | DESC], ... [WITH ROLLUP]]
      [HAVING where_condition]
      [ORDER BY {col_name | expr | position} [ASC | DESC], ...]
      [LIMIT {[offset,] row_count | row_count OFFSET offset  [ROWS EXAMINED rows_limit] } |
        [OFFSET start { ROW | ROWS }]
        [FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } { ONLY | WITH TIES }] ]
      procedure|[PROCEDURE procedure_name(argument_list)]
      [INTO OUTFILE 'file_name' [CHARACTER SET charset_name] [export_options] |
        INTO DUMPFILE 'file_name' | INTO var_name [, var_name] ]
      [FOR UPDATE lock_option | LOCK IN SHARE MODE lock_option]

export_options:
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
    ]

lock_option:
    [WAIT n | NOWAIT | SKIP LOCKED]
```

Description

`SELECT` is used to retrieve rows selected from one or more tables, and can include UNION statements and subqueries.

SELECT can also be used to retrieve rows computed without reference to any table.

Select Expressions

A `SELECT` statement must contain one or more select expressions, separated by commas. Each select expression can be one of the following:

When specifying a column, you can either use just the column name or qualify the column name with the name of the table using `tbl_name.col_name`. The qualified form is useful if you are joining multiple tables in the `FROM` clause. If you do not qualify the column names when selecting from multiple tables, MariaDB will try to find the column in each table. It is an error if that column name exists in multiple tables.

You can quote column names using backticks. If you are qualifying column names with table names, quote each part separately as `` `tbl_name`.`col_name` ``.

Contents

1. Syntax
2. Description
   1. Select Expressions
   2. DISTINCT
   3. INTO
   4. LIMIT
   5. LOCK IN SHARE MODE/FOR UPDATE
   6. OFFSET ... FETCH
   7. ORDER BY
   8. PARTITION
   9. PROCEDURE
   10. SKIP LOCKED
   11. SQL_CALC_FOUND_ROWS
   12. max_statement_time clause
   13. WAIT/NOWAIT
3. Examples
4. See Also

If you use any grouping functions in any of the select expressions, all rows in your results will be implicitly grouped, as if you had used `GROUP BY NULL`.

DISTINCT

A query may produce some identical rows. By default, all rows are retrieved, even when their values are the same. To explicitly specify that you want to retrieve identical rows, use the `ALL` option. If you want duplicates to be removed from the resultset, use the `DISTINCT` option. `DISTINCTROW` is a synonym for `DISTINCT`. See also COUNT DISTINCT and SELECT UNIQUE in Oracle mode.

INTO

The `INTO` clause is used to specify that the query results should be written to a file or variable.

The reverse of `SELECT INTO OUTFILE` is LOAD DATA.

LIMIT

Restricts the number of returned rows. See LIMIT and LIMIT ROWS EXAMINED for details.

LOCK IN SHARE MODE/FOR UPDATE

See LOCK IN SHARE MODE and FOR UPDATE for details on the respective locking clauses.

OFFSET ... FETCH

MariaDB starting with 10.6

See SELECT ... OFFSET ... FETCH.

ORDER BY

Order a resultset. See ORDER BY for details.

PARTITION

Specifies to the optimizer which partitions are relevant for the query. Other partitions will not be read. See Partition Pruning and Selection for details.

PROCEDURE

Passes the whole result set to a C Procedure. See PROCEDURE and PROCEDURE ANALYSE (the only built-in procedure not requiring the server to be recompiled).

SKIP LOCKED

MariaDB starting with 10.6

The SKIP LOCKED clause was introduced in MariaDB 10.6.0.

This causes those rows that couldn't be locked (LOCK IN SHARE MODE or FOR UPDATE) to be excluded from the result set. An explicit `NOWAIT` is implied here. This is only implemented on InnoDB tables and ignored otherwise.

SQL_CALC_FOUND_ROWS

When `SQL_CALC_FOUND_ROWS` is used, then MariaDB will calculate how many rows would have been in the result, if there would be no LIMIT clause. The result can be found by calling the function FOUND_ROWS() in your next sql statement.

max_statement_time clause

By using max_statement_time in conjunction with SET STATEMENT, it is possible to limit the execution time of individual queries. For example:

```
SET STATEMENT max_statement_time=100 FOR
  SELECT field1 FROM table_name ORDER BY field1;
```

WAIT/NOWAIT

Set the lock wait timeout. See WAIT and NOWAIT.

Examples

```
SELECT f1,f2 FROM t1 WHERE (f3<=10) AND (f4='y');
```

See Getting Data from MariaDB (Beginner tutorial), or the various sub-articles, for more examples.

LIMIT

Description

Use the `LIMIT` clause to restrict the number of returned rows. When you use a single integer *n* with `LIMIT`, the first *n* rows will be returned. Use the ORDER BY clause to control which rows come first. You can also select a number of rows after an offset using either of the following:

```
LIMIT offset, row_count
LIMIT row_count OFFSET offset
```

Contents

1. Description
   1. Multi-Table Updates
   2. GROUP_CONCAT
2. Examples
3. See Also

When you provide an offset *m* with a limit *n*, the first *m* rows will be ignored, and the following *n* rows will be returned.

Executing an UPDATE with the `LIMIT` clause is not safe for replication. `LIMIT 0` is an exception to this rule (see MDEV-6170).

There is a LIMIT ROWS EXAMINED optimization which provides the means to terminate the execution of SELECT statements which examine too many rows, and thus use too many resources. See LIMIT ROWS EXAMINED.

Multi-Table Updates

MariaDB starting with 10.3.2

Until MariaDB 10.3.1, it was not possible to use `LIMIT` (or ORDER BY) in a multi-table UPDATE statement. This restriction was lifted in MariaDB 10.3.2.

GROUP_CONCAT

MariaDB starting with 10.3.2

Starting from MariaDB 10.3.3, it is possible to use `LIMIT` with GROUP_CONCAT().

Examples

```
CREATE TABLE members (name VARCHAR(20));
INSERT INTO members VALUES('Jagdish'),('Kenny'),('Rokurou'),('Immaculada');

SELECT * FROM members;
+------------+
| name       |
+------------+
| Jagdish    |
| Kenny      |
| Rokurou    |
| Immaculada |
+------------+
```

Select the first two names (no ordering specified):

```
SELECT * FROM members LIMIT 2;
+---------+
| name    |
+---------+
| Jagdish |
| Kenny   |
+---------+
```

All the names in alphabetical order:

```
SELECT * FROM members ORDER BY name;
+------------+
| name       |
+------------+
| Immaculada |
| Jagdish    |
| Kenny      |
| Rokurou    |
+------------+
```

The first two names, ordered alphabetically:

```
SELECT * FROM members ORDER BY name LIMIT 2;
+------------+
| name       |
+------------+
| Immaculada |
| Jagdish    |
+------------+
```

The third name, ordered alphabetically (the first name would be offset zero, so the third is offset two):

```
SELECT * FROM members ORDER BY name LIMIT 2,1;
+-------+
| name  |
+-------+
| Kenny |
+-------+
```

From MariaDB 10.3.2, `LIMIT` can be used in a multi-table update:

```
CREATE TABLE warehouse (product_id INT, qty INT);
INSERT INTO warehouse VALUES (1,100),(2,100),(3,100),(4,100);

CREATE TABLE store (product_id INT, qty INT);
INSERT INTO store VALUES (1,5),(2,5),(3,5),(4,5);

UPDATE warehouse,store SET warehouse.qty = warehouse.qty-2, store.qty = store.qty+2
  WHERE (warehouse.product_id = store.product_id AND store.product_id  >= 1)
    ORDER BY store.product_id DESC LIMIT 2;

SELECT * FROM warehouse;
+------------+------+
| product_id | qty  |
+------------+------+
|          1 |  100 |
|          2 |  100 |
|          3 |   98 |
|          4 |   98 |
+------------+------+

SELECT * FROM store;
+------------+------+
| product_id | qty  |
+------------+------+
|          1 |    5 |
|          2 |    5 |
|          3 |    7 |
|          4 |    7 |
+------------+------+
```

From MariaDB 10.3.3, `LIMIT` can be used with GROUP_CONCAT, so, for example, given the following table:

```
CREATE TABLE d (dd DATE, cc INT);

INSERT INTO d VALUES ('2017-01-01',1);
INSERT INTO d VALUES ('2017-01-02',2);
INSERT INTO d VALUES ('2017-01-04',3);
```

the following query:

```
SELECT SUBSTRING_INDEX(GROUP_CONCAT(CONCAT_WS(":",dd,cc) ORDER BY cc DESC),",",1) FROM d;
+----------------------------------------------------------------------------+
| SUBSTRING_INDEX(GROUP_CONCAT(CONCAT_WS(":",dd,cc) ORDER BY cc DESC),",",1) |
+----------------------------------------------------------------------------+
| 2017-01-04:3                                                               |
+----------------------------------------------------------------------------+
```

can be more simply rewritten as:

```
SELECT GROUP_CONCAT(CONCAT_WS(":",dd,cc) ORDER BY cc DESC LIMIT 1) FROM d;
+-------------------------------------------------------------+
| GROUP_CONCAT(CONCAT_WS(":",dd,cc) ORDER BY cc DESC LIMIT 1) |
+-------------------------------------------------------------+
| 2017-01-04:3                                                |
+-------------------------------------------------------------+
```

ORDER BY

Description

Contents

1. Description
2. Examples
3. See Also

Use the `ORDER BY` clause to order a resultset, such as that are returned from a SELECT statement. You can specify just a column or use any expression with functions. If you are using the `GROUP BY` clause, you can use grouping functions in `ORDER BY`. Ordering is done after grouping.

You can use multiple ordering expressions, separated by commas. Rows will be sorted by the first expression, then by the second expression if they have the same value for the first, and so on.

You can use the keywords `ASC` and `DESC` after each ordering expression to force that ordering to be ascending or descending, respectively. Ordering is ascending by default.

You can also use a single integer as the ordering expression. If you use an integer *n*, the results will be ordered by the *n*th column in the select expression.

When string values are compared, they are compared as if by the STRCMP function. `STRCMP` ignores trailing whitespace and may normalize characters and ignore case, depending on the collation in use.

Duplicated entries in the `ORDER BY` clause are removed.

`ORDER BY` can also be used to order the activities of a DELETE or UPDATE statement (usually with the LIMIT clause).

MariaDB starting with 10.3.2

Until MariaDB 10.3.1, it was not possible to use `ORDER BY` (or LIMIT) in a multi-table UPDATE statement. This restriction was lifted in MariaDB 10.3.2.

MariaDB starting with 10.5

From MariaDB 10.5, MariaDB allows packed sort keys and values of non-sorted fields in the sort buffer. This can make filesort temporary files much smaller when VARCHAR, CHAR or BLOBs are used, notably speeding up some ORDER BY sorts.

Examples

```
CREATE TABLE seq (i INT, x VARCHAR(1));
INSERT INTO seq VALUES (1,'a'), (2,'b'), (3,'b'), (4,'f'), (5,'e');

SELECT * FROM seq ORDER BY i;
+------+------+
| i    | x    |
+------+------+
|    1 | a    |
|    2 | b    |
|    3 | b    |
|    4 | f    |
|    5 | e    |
+------+------+

SELECT * FROM seq ORDER BY i DESC;
+------+------+
| i    | x    |
+------+------+
|    5 | e    |
|    4 | f    |
|    3 | b    |
|    2 | b    |
|    1 | a    |
+------+------+

SELECT * FROM seq ORDER BY x,i;
+------+------+
| i    | x    |
+------+------+
|    1 | a    |
|    2 | b    |
|    3 | b    |
|    5 | e    |
|    4 | f    |
+------+------+
```

ORDER BY in an UPDATE statement, in conjunction with LIMIT:

```
UPDATE seq SET x='z' WHERE x='b' ORDER BY i DESC LIMIT 1;

SELECT * FROM seq;
+------+------+
| i    | x    |
+------+------+
|    1 | a    |
|    2 | b    |
|    3 | z    |
|    4 | f    |
|    5 | e    |
+------+------+
```

From MariaDB 10.3.2, `ORDER BY` can be used in a multi-table update:

```
CREATE TABLE warehouse (product_id INT, qty INT);
INSERT INTO warehouse VALUES (1,100),(2,100),(3,100),(4,100);

CREATE TABLE store (product_id INT, qty INT);
INSERT INTO store VALUES (1,5),(2,5),(3,5),(4,5);

UPDATE warehouse,store SET warehouse.qty = warehouse.qty-2, store.qty = store.qty+2
  WHERE (warehouse.product_id = store.product_id AND store.product_id  >= 1)
    ORDER BY store.product_id DESC LIMIT 2;

SELECT * FROM warehouse;
+------------+------+
| product_id | qty  |
+------------+------+
|          1 |  100 |
|          2 |  100 |
|          3 |   98 |
|          4 |   98 |
+------------+------+

SELECT * FROM store;
+------------+------+
| product_id | qty  |
+------------+------+
|          1 |    5 |
|          2 |    5 |
|          3 |    7 |
|          4 |    7 |
+------------+------+
```

UNION

`UNION` is used to combine the results from multiple SELECT statements into a single result set.

Syntax

```
SELECT ...
UNION [ALL | DISTINCT] SELECT ...
[UNION [ALL | DISTINCT] SELECT ...]
[ORDER BY [column [, column ...]]]
[LIMIT {[offset,] row_count | row_count OFFSET offset}]
```

Description

`UNION` is used to combine the results from multiple SELECT statements into a single result set.

The column names from the first `SELECT` statement are used as the column names for the results returned. Selected columns listed in corresponding positions of each SELECT statement should have the same data type. (For example, the first column selected by the first statement should have the same type as the first column selected by the other statements.)

If they don't, the type and length of the columns in the result take into account the values returned by all of the SELECTs, so there is no need for explicit casting. Note that currently this is not the case for recursive CTEs - see MDEV-12325.

Contents

1. Syntax
2. Description
   1. ALL/DISTINCT
   2. ORDER BY and LIMIT
   3. HIGH_PRIORITY
   4. SELECT ... INTO ...
   5. Parentheses
3. Examples
4. See Also

Table names can be specified as `db_name` . `tbl_name` . This permits writing `UNION` s which involve multiple databases. See Identifier Qualifiers for syntax details.

UNION queries cannot be used with aggregate functions.

`EXCEPT` and `UNION` have the same operation precedence and `INTERSECT` has a higher precedence, unless running in Oracle mode, in which case all three have the same precedence.

ALL/DISTINCT

The `ALL` keyword causes duplicate rows to be preserved. The `DISTINCT` keyword (the default if the keyword is omitted) causes duplicate rows to be removed by the results.

UNION ALL and UNION DISTINCT can both be present in a query. In this case, UNION DISTINCT will override any UNION ALLs to its left.

MariaDB starting with 10.1.1

Until MariaDB 10.1.1, all `UNION ALL` statements required the server to create a temporary table. Since MariaDB 10.1.1, the server can in most cases execute `UNION ALL` without creating a temporary table, improving performance (see MDEV-334).

ORDER BY and LIMIT

Individual SELECTs can contain their own ORDER BY and LIMIT clauses. In this case, the individual queries need to be wrapped between parentheses. However, this does not affect the order of the UNION, so they only are useful to limit the record read by one SELECT.

The UNION can have global ORDER BY and LIMIT clauses, which affect the whole resultset. If the columns retrieved by individual SELECT statements have an alias (AS), the ORDER BY must use that alias, not the real column names.

HIGH_PRIORITY

Specifying a query as HIGH_PRIORITY will not work inside a UNION. If applied to the first SELECT, it will be ignored. Applying to a later SELECT results in a syntax error:

```
ERROR 1234 (42000): Incorrect usage/placement of 'HIGH_PRIORITY'
```

SELECT ... INTO ...

Individual SELECTs cannot be written INTO DUMPFILE or INTO OUTFILE. If the last SELECT statement specifies INTO DUMPFILE or INTO OUTFILE, the entire result of the UNION will be written. Placing the clause after any other SELECT will result in a syntax error.

If the result is a single row, SELECT ... INTO @var_name can also be used.

MariaDB starting with 10.4.0

Parentheses

From MariaDB 10.4.0, parentheses can be used to specify precedence. Before this, a syntax error would be returned.

Examples

`UNION` between tables having different column names:

```
(SELECT e_name AS name, email FROM employees)
UNION
(SELECT c_name AS name, email FROM customers);
```

Specifying the `UNION` 's global order and limiting total rows:

```
(SELECT name, email FROM employees)
UNION
(SELECT name, email FROM customers)
ORDER BY name LIMIT 10;
```

Adding a constant row:

```
(SELECT 'John Doe' AS name, 'john.doe@example.net' AS email)
UNION
(SELECT name, email FROM customers);
```

Differing types:

```
SELECT CAST('x' AS CHAR(1)) UNION SELECT REPEAT('y',4);
+----------------------+
| CAST('x' AS CHAR(1)) |
+----------------------+
| x                    |
| yyyy                 |
+----------------------+
```

Returning the results in order of each individual SELECT by use of a sort column:

```
(SELECT 1 AS sort_column, e_name AS name, email FROM employees)
UNION
(SELECT 2, c_name AS name, email FROM customers) ORDER BY sort_column;
```

Difference between UNION, EXCEPT and INTERSECT. `INTERSECT ALL` and `EXCEPT ALL` are available from MariaDB 10.5.0.

```
CREATE TABLE seqs (i INT);
INSERT INTO seqs VALUES (1),(2),(2),(3),(3),(4),(5),(6);

SELECT i FROM seqs WHERE i <= 3 UNION SELECT i FROM seqs WHERE i>=3;
+------+
| i    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    5 |
|    6 |
+------+

SELECT i FROM seqs WHERE i <= 3 UNION ALL SELECT i FROM seqs WHERE i>=3;
+------+
| i    |
+------+
|    1 |
|    2 |
|    2 |
|    3 |
|    3 |
|    3 |
|    3 |
|    4 |
|    5 |
|    6 |
+------+

SELECT i FROM seqs WHERE i <= 3 EXCEPT SELECT i FROM seqs WHERE i>=3;
+------+
| i    |
+------+
|    1 |
|    2 |
+------+

SELECT i FROM seqs WHERE i <= 3 EXCEPT ALL SELECT i FROM seqs WHERE i>=3;
+------+
| i    |
+------+
|    1 |
|    2 |
|    2 |
+------+

SELECT i FROM seqs WHERE i <= 3 INTERSECT SELECT i FROM seqs WHERE i>=3;
+------+
| i    |
+------+
|    3 |
+------+

SELECT i FROM seqs WHERE i <= 3 INTERSECT ALL SELECT i FROM seqs WHERE i>=3;
+------+
| i    |
+------+
|    3 |
|    3 |
+------+
```

Parentheses for specifying precedence, from MariaDB 10.4.0

```
CREATE OR REPLACE TABLE t1 (a INT);
CREATE OR REPLACE TABLE t2 (b INT);
CREATE OR REPLACE TABLE t3 (c INT);

INSERT INTO t1 VALUES (1),(2),(3),(4);
INSERT INTO t2 VALUES (5),(6);
INSERT INTO t3 VALUES (1),(6);

((SELECT a FROM t1) UNION (SELECT b FROM t2)) INTERSECT (SELECT c FROM t3);
+------+
| a    |
+------+
|    1 |
|    6 |
+------+

(SELECT a FROM t1) UNION ((SELECT b FROM t2) INTERSECT (SELECT c FROM t3));
+------+
| a    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    6 |
+------+
```