What's new in MariaDB 10.3

Shenzhen, Nov 2017

Michael "Monty" Widenius
Entrepreneur, MariaDB hacker
monty@mariadb.org
http://mariadb.org/
http://mariadb.com/

Introducing Maria (DB)

Introducing Maria

*MariaDB **server** releases*

- MariaDB 5.1 (Feb 2010) - Making builds free
- MariaDB 5.2 (Nov 2010) - Community features
- MariaDB 5.3 (Apr 2012) - New optimizer
- MariaDB 5.5 (Apr 2013) - Merge with MySQL
- MariaDB 10.0 (Mar 2014) - Parallel replication
- MariaDB 10.1 (Oct 2015) - Galera, Encryption
- MariaDB 10.2 (Apr 2017) - Advanced features
- MariaDB 10.3 (RC Dec 2017) - Compatibility

SEQUENCE

CREATE [OR REPLACE] [TEMPORARY] **SEQUENCE** [IF NOT EXISTS] sequence_name

[INCREMENT [BY | =] increment]

[MINVALUE [=] minvalue | NO MINVALUE | NOMINVALUE]

[MAXVALUE [=] maxvalue | NO MAXVALUE | NOMAXVALUE]

[START [WITH | =] start]

[CACHE [=] cache] [[NO] CYCLE] [table_options]

SEQUENCE

CREATE SEQUENCE s1 START WITH 50 MAXVALUE 1000;

SHOW CREATE SEQUENCE s1;

CREATE SEQUENCE `s1` start with 50 minvalue 1 maxvalue 5000 increment by 1 cache 1000 nocycle ENGINE=Aria

*SELECT * FROM s\g*

next_not_cached_value: 50
minimum_value: 1
maximum_value: 5000
start_value: 50
increment: 1
cache_size: 1000
cycle_option: 0
cycle_count: 0

SEQUENCE

- **ALTER SEQUENCE**
- **DROP SEQUENCE**
- **NEXT VALUE FOR** sequence_name
  - **NEXTVAL**(sequence_name) and sequence_name.**nextval** are supported
- **PREVIOUS VALUE FOR** sequence_name
  - **LASTVAL**(sequence_name) and sequence_name.**currval** are supported
- **SETVAL**(sequence_name, next_value, [is_used, [round]])
  - SELECT SETVAL('foo', 42); -- Next **NEXTVAL()** will return 43
  - SELECT SETVAL('foo', 42, true); -- Same as above
  - SELECT SETVAL('foo', 42, false); -- Next **NEXTVAL()** will return 42
  - SETVAL() will not set the SEQUENCE value to a something that is less than its current value. This is to make SETVAL() replication safe.

INTERSECT & EXCEPT

- (select a,b from t1) **INTERSECT** (select c,d from t2)
  - Returns all rows that are in both SELECT

- (select a,b from t1) **EXCEPT** (select c,d from t2)
  - Returns rows that are in first SELECT but not in second SELECT

ROW data type for stored routine variables

ROW data type

<row type> ::= ROW <row type body>

<row type body> ::= <left paren> <field definition> [{ <comma> <field definition> }...] <right paren>

<field definition> ::= <field name> <data type>

<data type> ::= <predefined type>

```
CREATE PROCEDURE p1()
BEGIN
  DECLARE a ROW (c1 INT, c2 VARCHAR(10));
  SET a.c1= 10;
  SET a.c2= 'test';
  INSERT INTO t1 VALUES (a.c1, a.c2);
END;
CALL p1();
```

TYPE OF and ROW TYPE OF

- DECLARE tmp TYPE OF t1.a;
  - Get the data type from the column **a** in the table **t1**
- DECLARE rec1 ROW TYPE OF t1;
  - Get the row data type from the table **t1**
- DECLARE rec2 ROW TYPE OF cur1;
  - Get the row data type from the cursor **cur1**

Cursors with parameters

DECLARE cursor_name CURSOR FOR select_statement;
OPEN cursor_name;

Is extended with parameters:

DECLARE cursor_name CURSOR
[(cursor_formal_parameter[,...]]) FOR select_statement;
OPEN cursor_name [(expression[,...]]);
cursor_formal_parameter:
  name type [collate clause]

DECLARE cur CURSOR(**pmin** INT, **pmax** INT) FOR SELECT a FROM t1 WHERE a BETWEEN **pmin** AND **pmax**;

Compatibility parser
Cursors with parameters

```
SET sql_mode=ORACLE;
DELIMITER $$
CREATE OR REPLACE PROCEDURE p1 AS
  CURSOR c(prm_a VARCHAR2, prm_b VARCHAR2) IS
    SELECT a, b, c FROM t1 WHERE a=prm_a AND b=prm_b;
  v_a INT;
  v_b INT;
  v_c INT;
BEGIN
  OPEN c(1, 2);
  FETCH c INTO v_a, v_b, v_c;
  CLOSE c;
  INSERT INTO t1 VALUES (v_a + 10, v_b + 10, v_c + 10);
END;
$$
```

Compatibility parser
Cursors with parameters + FOR

```
SET sql_mode=ORACLE;
DELIMITER $$
CREATE OR REPLACE PROCEDURE p1 AS
  CURSOR c(prm_a VARCHAR2, prm_b VARCHAR2) IS
    SELECT a, b, c FROM t1 WHERE a=prm_a AND b=prm_b;
BEGIN
  FOR rec IN c(1, 2)
  LOOP
    INSERT INTO t2 VALUES (rec.a, rec.b, rec.c);
  END LOOP;
END;
$$
DELIMITER ;
```

WAIT and NO_WAIT

The feature provides a way to set **lock wait timeout** for some statement.

- select ... for update [wait [n] | no_wait]
- select ... lock in share mode [wait [n] | no_wait]
- lock table ... [wait [n] | no_wait]
- alter table t [wait [n] | no_wait] add f int,
- drop table [wait [n] | no_wait]

Extending DELETE

- DELETE statement can delete from the table that is used in a subquery in the WHERE clause

```
DELETE FROM t1 WHERE c1 IN (SELECT b.c1 FROM t1 b
WHERE b.c2=0)
```

Extending UPDATE

- UPDATE statements can use same source and target

```
UPDATE t1 SET c1=c1+1 WHERE c2=(SELECT MAX(c2) FROM t1);
```

- MULTI-TABLE UPDATE now supports ORDER BY and LIMIT

```
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;
```

Instant ADD COLUMN for InnoDB

Adding new columns last to an InnoDB table is “Instant” thanks to a new (default) table format in 10.3

Limitations:

- New column(s) must be last
- No FULLTEXT INDEX
- If the table becomes empty (either via TRUNCATE or DELETE), the table will be converted to the old "non-instant" format.
- InnoDB data files after instant ADD COLUMN cannot be imported to older versions of MariaDB or MySQL.
- Instant ADD COLUMN is not available for ROW_FORMAT=COMPRESSED.
- ALTER TABLE…DROP COLUMN requires the table to be rebuilt.

Compressed columns

New COMPRESSED column attribute for CREATE TABLE:
COMPRESSED[=<compression_method>]

CREATE TABLE cmp (i TEXT COMPRESSED);
CREATE TABLE cmp2 (i TEXT COMPRESSED=zlib);

- Works for all storage engines
- Compression only done when accessing value, not when internally copying values between tables.
- Currently only supported method is zlib
- Not possible to create indexes on compressed columns
- CSV engine automatically decompresses on storage

New Functionallity

- CHR(#) - Returns VARCHAR(1) of the corresponding character
- DATE_FORMAT(date, format, locale) with 3 arguments
  - Locale makes DATE_FORMAT() session independent
- The server now supports the PROXY protocol
  - https://www.haproxy.org/download/1.8/doc/proxy-protocol.txt
- Temporary files created by merge sort and row log are always encrypted if **innodb_encrypt_log** is set to 1
- Condition pushdown through PARTITION BY clause of window functions
- New optimizer switch setting, split_grouping_derived
- New system variable gtid_pos_auto_engines for improving performance if a server is using multiple different storage engines in different transactions

New Functionallity

- Disable logging of certain statements to the general log or the slow query log with the log_disabled_statements and log_slow_disabled_statements system variables.
- Connections with idle transactions can be automatically killed after a specified time period by means of the idle_transaction_timeout, idle_readonly_transaction_timeout and idle_write_transaction_timeout system variables.
- More server states gives more accurate profiling of where mysqld is spending time.

New “Oracle compatible” parser

Providing compatibility for basic PL/SQL constructs with new parser

```
SET SQL_MODE=ORACLE;

CREATE PROCEDURE sp1 (p1 IN VARCHAR2, p2 OUT VARCHAR2)
IS
  v1 VARCHAR2(100);
BEGIN
  v1 := p1;
  p2 := v1;
END;
```

Compatibility parser
Label and IN/OUT

MariaDB syntax:

```
label:
```

Oracle syntax:

```
<<label>>
```

Different order of IN, OUT, INOUT keywords

MariaDB:

```
CREATE PROCEDURE p1 (OUT param INT)
```

Oracle:

```
CREATE PROCEDURE p1 (param OUT INT)
```

Compatibility parser
GOTO statement

```
SET sql_mode=ORACLE;
DELIMITER $$
CREATE OR REPLACE PROCEDURE p1(a INT) AS
BEGIN
  IF (a < 0) THEN
    GOTO ret;
  END IF;
  INSERT INTO log (msg) VALUES ('a is negative');
<<ret>>
  INSERT INTO t1 (a) VALUES (10);
END;
$$
```

Compatibility parser
AS and IS

Oracle requires AS or IS keyword before the body:

```
CREATE FUNCTION f1 RETURN NUMBER
AS
BEGIN
   RETURN 10;
END;
```

Compatibility parser
EXIT statement

```
MariaDB syntax:
IF bool_expr THEN LEAVE label

Oracle syntax to leave a loop block:
EXIT [ label ] [ WHEN bool_expr ]
```

Compatibility parser
Assignment operator

```
MariaDB syntax:
SET var= 10;

Oracle:
var:= 10;

Assignment operator works for
MariaDB system variables:

max_sort_length:=1025;
```

Compatibility parser DECLARE

MariaDB syntax:

```
BEGIN
  DECLARE a INT;
  DECLARE b VARCHAR(10);
  v= 10;
END;
```

Oracle syntax (declaration after DECLARE or AS):

```
DECLARE
  a INT;
  b VARCHAR(10);
BEGIN
  v:= 10;
END;
```

Compatibility parser
EXCEPTION handlers

MariaDB uses DECLARE HANDLER to catch exceptions:

```
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
  ..
  END;
END;
```

In Oracle, exception handlers are declared in the end of a block:

```
BEGIN
  ...
EXCEPTION
  WHEN OTHERS THEN
  BEGIN ..
  END;
END;
```

Compatibility parser

Oracle predefined exceptions

Three most important Oracle pre-defined exceptions implemented:

- TOO_MANY_ROWS THEN
- NO_DATA_FOUND THEN
- DUP_VAL_ON_INDEX

Can be used in both RAISE and EXCEPTION..WHEN:

```
CREATE OR REPLACE PROCEDURE p1
BEGIN
   ...
   RAISE TOO_MANY_ROWS;
   ...
EXCEPTION
   WHEN TOO_MANY_ROWS THEN ...
END;
```

Compatibility parser

User-defined exceptions

```
CREATE OR REPLACE FUNCTION f1 (a INT) RETURN INT
AS
  e1 EXCEPTION;
BEGIN
  IF a < 0 THEN
    RAISE e1;
  END IF;
  RETURN 0;
EXCEPTION
  WHEN e1 THEN RETURN 1;
END;
```

Compatibility parser

New Syntax

```
Default variable value:
x INT := 10;

NULL as a statement:
CREATE PROCEDURE p1() AS
BEGIN
  NULL;
END;

NULL can appear in other syntactic
constructs as a statement:
 IF a=10 THEN NULL; ELSE NULL; END IF;
```

Compatibility parser

Function parameters

If no parameters, then parentheses must be omitted:

```
CREATE PROCEDURE p1 AS
BEGIN
  NULL;
END;
```

IN OUT instead of INOUT

```
CREATE PROCEDURE p1 (a IN OUT INT)
AS
BEGIN
END;
```

Compatibility parser

ELSIF vs ELSEIF

Oracle uses ELSIF:

```
CREATE FUNCTION f1(a INT) RETURN VARCHAR
AS
BEGIN
  IF a=1 THEN RETURN 'a is 1';
  ELSIF a=2 THEN RETURN 'a is 2';
  ELSE RETURN 'a is unknown';
  END IF;
END;
```

Compatibility parser
RETURN

Oracle uses RETURN rather than RETURNS:
CREATE FUNCTION f1(a INT) RETURN INT …

MariaDB supports RETURNS only in stored functions.
Oracle supports RETURN in stored procedures as well

```
CREATE PROCEDURE p1 (a IN OUT INT)
AS
BEGIN
  IF a < 10 THEN
    RETURN;
  END IF;
  a:=a+1;
EXCEPTION
  WHEN OTHERS THEN RETURN;
END;
```

Compatibility parser
WHILE and CONTINUE

MariaDB syntax:
[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]

Oracle syntax:
[<<label>>]
WHILE boolean_expression
  LOOP statement... END LOOP [label] ;

CONTINUE statement:
CONTINUE [label] [WHEN boolean_expression] ;

This is a replacement for the ITERATE statement in MariaDB.
CONTINUE is valid only inside a LOOP.

Compatibility parser
Dynamic SQL placeholders

MariaDB syntax:

```
SET sql_mode=DEFAULT;
PREPARE stmt FROM 'SELECT ?, ?';
EXECUTE stmt USING @10, @20;
DEALLOCATE PREPARE stmt;
```

Oracle-style syntax:

```
SET sql_mode=ORACLE;
PREPARE stmt FROM 'SELECT :1, :2';
EXECUTE stmt USING @10, @20;
DEALLOCATE PREPARE stmt;
```

Compatibility parser
Numeric FOR loop

```
SET sql_mode=ORACLE;
DELIMITER $$
CREATE OR REPLACE PROCEDURE p1 AS
BEGIN
  FOR i IN 1..10 LOOP
    SELECT i;
  END LOOP;
END;
$$
```

Compatibility parser
Explicit cursor FOR loop

```
SET sql_mode=ORACLE;
DELIMITER $$
CREATE OR REPLACE PROCEDURE p1 AS
  CURSOR cur IS (SELECT a, b FROM t1);
BEGIN
  FOR rec IN cur
  LOOP
    SELECT rec.a, rec.b;
  END LOOP;
END;
$$
```

Compatibility parser
Implicit cursor FOR loop

```
SET sql_mode=ORACLE;
DELIMITER $$
CREATE OR REPLACE PROCEDURE p1 AS
BEGIN
  FOR rec IN (SELECT a, b FROM t1)
  LOOP
    SELECT rec.a, rec.b;
  END LOOP;
END;
$$
```

Compatibility parser
Anchored data types

```
SET sql_mode=ORACLE;
DELIMITER $$
CREATE OR REPLACE PROCEDURE p1
AS
  va t1.c1%TYPE; - take data type from a table column
  vb va%TYPE; - take data type from another variable
BEGIN
  SELECT MAX(a),MIN(a) INTO va,vb FROM t1;
  INSERT INTO t2 VALUES (va),(vb);
END;
$$
DELIMITER ;
```

Compatibility parser
Anchored table ROW types

```
SET sql_mode=ORACLE;
DELIMITER $$
CREATE OR REPLACE PROCEDURE p1
AS
  rec1 t1%ROWTYPE; -- take ROW structure from a table
BEGIN
  SELECT * FROM t1 INTO rec1;
  INSERT INTO t2 VALUES (rec1.a, rec1.b);
END;
$$
DELIMITER ;
```

Compatibility parser
Anchored cursor ROW types

```
SET sql_mode=ORACLE;
DELIMITER $$
CREATE OR REPLACE PROCEDURE p1
AS
  CURSOR c1 IS SELECT a,b FROM t1;
  rec1 c1%ROWTYPE; -- take ROW structure from a cursor
BEGIN
  OPEN c1;
  FETCH c1 INTO rec1;
  CLOSE c1;
  INSERT INTO t2 VALUES (rec1.a, rec1.b);
END;
$$
DELIMITER ;
```

Compatibility parser
Anchors to ROW variables

```
SET sql_mode=ORACLE;
DELIMITER $$
CREATE OR REPLACE PROCEDURE p1
AS
  CURSOR c1 IS SELECT a,b FROM t1;
  rec1 c1%ROWTYPE;
  rec2 t1%ROWTYPE;
  rec3 ROW(a INT,b TEXT);
  rec11 rec1%TYPE; -- anchor to a table ROW variable
  rec12 rec2%TYPE; -- anchor to a cursor ROW variable
  rec31 rec3%TYPE; -- anchor to a explicit ROW variable
BEGIN
  /* … some code … */
END;
$$
DELIMITER ;
```

Compatibility parser
:NEW and :OLD vs NEW and OLD

```
SET sql_mode=ORACLE;
DELIMITER $$
CREATE OR REPLACE TABLE t1 (a INT, c INT);
CREATE OR REPLACE TRIGGER tr1 BEFORE INSERT ON t1
FOR EACH ROW
DECLARE
  cnt INT := 0;
BEGIN
  IF :NEW.a IS NULL THEN cnt:=cnt+1; END IF;
  IF :NEW.c IS NULL THEN :NEW.c:=cnt; END IF;
END;
$$
```

Compatibility parser

Compound statements (aka anonymous blocks)

MariaDB syntax:

```
BEGIN NOT ATOMIC
  SELECT 1;
END;
```

```
BEGIN NOT ATOMIC
  DECLARE a INT DEFAULT 1;
  DECLARE b TEXT DEFAULT 'b';
  SELECT a, b;
END;
```

Oracle syntax:

```
BEGIN
  SELECT 1;
END;
```

```
DECLARE
  a INT := 1;
  b TEXT := 'b';
BEGIN
  SELECT a, b;
END;
```

Compatibility parser Miscellaneous

1. UNIQUE as DISTINCT

```
SET sql_mode=ORACLE;
SELECT UNIQUE a,b FROM t1;
```

2. Translating empty string literals to NULL

```
SET sql_mode=EMPTY_STRING_IS_NULL;
SELECT '' IS NULL; -- returns TRUE
INSERT INTO t1 VALUES (''); -- inserts NULL
```

(more empty string sql_mode flags are coming soon)

Compatibility parser

Smaller sql_mode=ORACLE tasks

Data types:

- MDEV-10343 Providing compatibility for basic SQL data types
- MDEV-10596 Allow VARCHAR and VARCHAR2 without length as a data type of routine parameters and in RETURN clause
- MDEV-13919 Derive length of VARCHAR SP parameters with no length from actual parameters
- MDEV-11275 CAST(..AS VARCHAR(N))

SP syntax:

- MDEV-10598 Variable declarations can go after cursor declarations
- MDEV-10578 SP control functions SQLCODE, SQLERRM
- MDEV-12089 Understand optional routine name after the END keyword
- MDEV-12107 Inside routines the CALL keywoard is optional

Built-in functions and operators:

- MDEV-12783 Functions LENGTH() and LENGTHB()
- MDEV-11880 Make the concatenation operator ignore NULL arguments
- MDEV-12143 Make the CONCAT function ignore NULL arguments
- MDEV-12685 Oracle-compatible function CHR()
- MDEV-14012 substr(): treat position 0 as position 1

New syntax
ROW as a routine parameter

```
DELIMITER $$
CREATE OR REPLACE
      PROCEDURE p1(OUT a ROW(a INT, b TEXT),
                   IN  b ROW(a INT, b TEXT))
BEGIN
  SET a.a=a.a+b.a;
  SET a.b=CONCAT(a.b,b.b);
END;
$$
```

New syntax
ROW in FETCH

```
DELIMITER $$
BEGIN NOT ATOMIC
  DECLARE row ROW (c1 INT, c2 VARCHAR(10));
  DECLARE cur CURSOR FOR SELECT 1, 'test';
  OPEN cur;
  FETCH cur INTO row;
  CLOSE cur;
  SELECT row.c1, row.c2;
END;
$$
DELIMITER ;
```

New syntax
ROW in SELECT INTO

```
DELIMITER $$
BEGIN NOT ATOMIC
  DECLARE row ROW (c1 INT, c2 VARCHAR(10));
  SELECT 1, 'test' INTO row;
  SELECT row.c1, row.c2;
END;
$$
DELIMITER ;
```

New syntax
NO PAD and partitions

MDEV-9711 NO PAD collations:

```
CREATE TABLE t1 (a VARCHAR(5)) DEFALT CHARSET utf8 COLLATE
utf8_general_nopad_ci;
INSERT INTO t1 VALUES ('a '),('a     ');
SELECT concat("*",a,"*") FROM t1 WHERE a='a ';
*A *
```

MDEV-8348 Add catchall to all table partitioning for list partitions:

```
PARTITION BY LIST (partitioning_expression)
(
        PARTITION partition_name VALUES IN (value_list),
        [ PARTITION partition_name VALUES IN (value_list), ... ]
       [ PARTITION partition_name DEFAULT ]
)
```

Compatibility for basic data types

- VARCHAR2 - a synonym to VARCHAR
- NUMBER - a synonym to DECIMAL
- DATE (with time portion) - a synonym to DATETIME
- RAW - a synonym to VARBINARY
- CLOB - a synonym to LONGTEXT
- BLOB - a synonym to LONGBLOB

Compatibility parser

- Explicit cursor attributes %ISOPEN, %ROWCOUNT, %FOUND, %NOTFOUND SQL%ROWCOUNT
- Variable declarations can go after cursor declarations
- Predefined exceptions: TOO_MANY_ROWS, NO_DATA_FOUND, DUP_VAL_ON_INDEX
- RAISE statement for predefined exceptions

Compatibility parser

- SP control functions SQLCODE, SQLERRM
- Allow VARCHAR and VARCHAR2 without length as a data type of routine parameters and in RETURN clause
- Anonymous blocks
- Do not require BEGIN..END in multi-statement exception handlers in THEN clause

Compatibility parser

- Understand optional routine name after the END keyword
- Inside routines the CALL keywoard is optional
- Make the concatenation functions ignore NULL arguments
- TRUNCATE TABLE t1 [{DROP|REUSE} STORAGE

Features that will be in 10.3
Packages

A package is a schema object that groups logically related PL/SQL data types, items (e.g. variables, cursors, exceptions) and subprograms.

- CREATE PACKAGE
- CREATE PACKAGE BODY
- ALTER PACKAGE
- DROP PACKAGE BODY
- DROP PACKAGE

Features that will be in 10.3
Hidden columns

1) **Not hidden** — normal columns created by the user
2) **A little bit hidden** — columns that the user has marked hidden. They aren't shown in SELECT * and they don't require values in INSERT table VALUE (...).
3) **More hidden** — Can be queried explicitly, otherwise hidden from everything, inclusive CREATE and ALTER. Think ROWID pseudo-column.
4) **Very hidden** — as above, but cannot be queried either. They can only show up in EXPLAIN EXTENDED. May be indexed.

Features that will be in 10.3
AS OF

- CREATE TABLE t1 (...) with system versioning
  - One can also specific versioning per column
- ALTER TABLE t1 WITH SYSTEM VERSIONING
- SELECT * FROM employee
  FOR SYSTEM_TIME
  AS OF TIMESTAMP '2016-02-06 10:03:03'
  WHERE name = 'john';

First version will only support DML (DROP TABLE will remove history), second version will also support DDL.

Features that will be in 10.3

- SPIDER updated to latest release
  - Now in bb-10.2-spider tree

Features that may be in 10.3

- Galera 4

Questions ?

For questions later, use the public MariaDB email list at maria-discuss@lists.launchpad.net or #maria on Freenode.