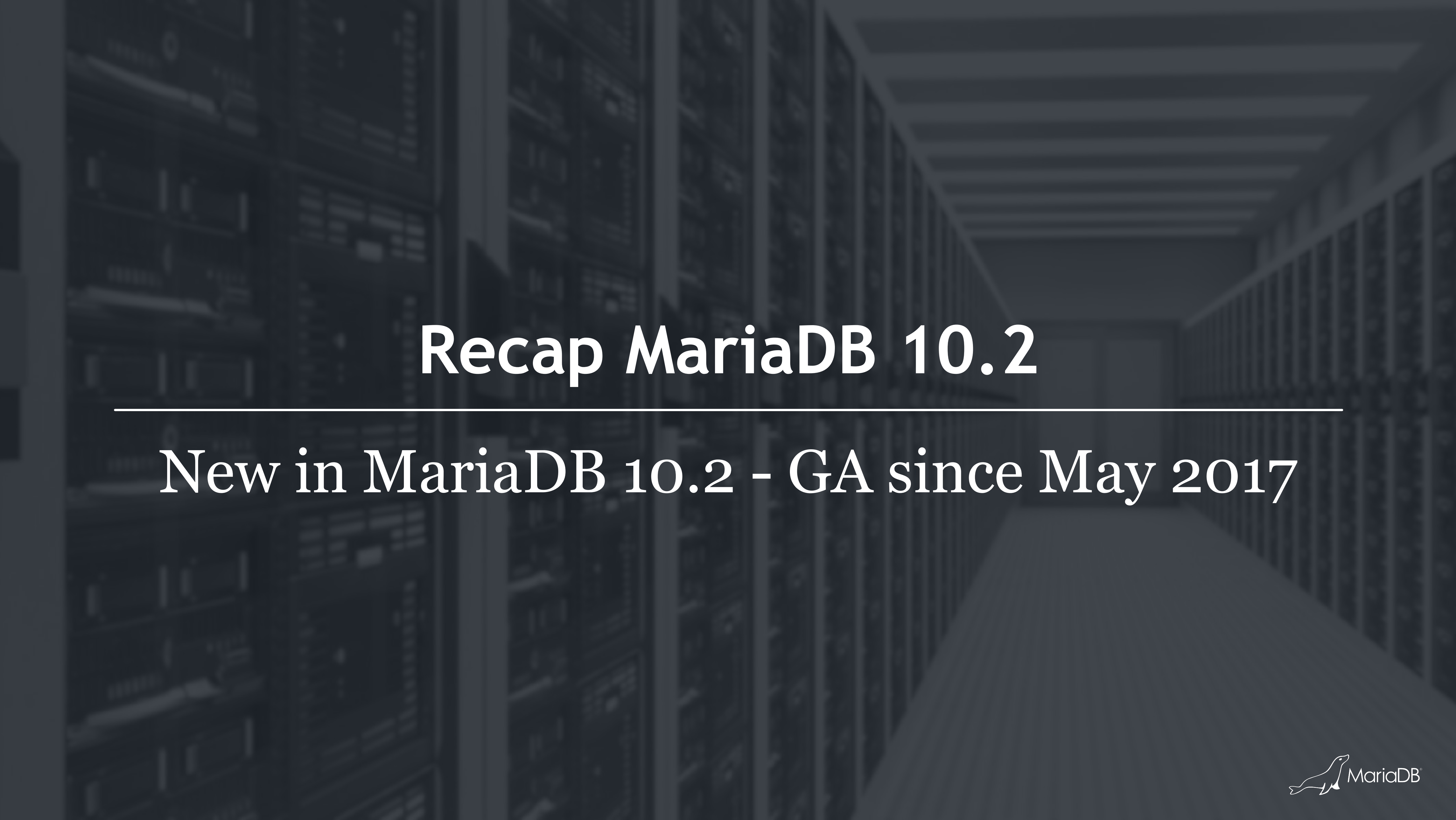
# What’s New in MariaDB Server 10.3

Max Mether
VP Server



# Recap MariaDB 10.2

New in MariaDB 10.2 - GA since May 2017

# What’s New in 10.2

| | |
|---|---|
| **Analytics SQL** | ● Window Functions<br>● Common Table Expressions (CTE) |
| **JSON** | ● JSON Functions<br>● GeoJSON Functions |
| **Replication** | ● Delayed Replication<br>● Restrict the speed of reading binlog from Master<br>● Compressed Binary Log |
| **Database Compatibility** | ● Multi-Trigger Support<br>● CHECK Constraint Expression Support<br>● EXECUTE IMMEDIATE statement<br>● Support for DEFAULT with expressions<br>● DECIMAL increased from 30 to 38 digits |
| **Storage Engine Enhancements** | ● New Storage Engine MyRocks based on RocksDB from Facebook<br>● Enhancements from MySQL InnoDB 5.7<br>● Enable InnoDB NUMA interleave for InnoDB |

# What’s New in 10.2

| | |
|---|---|
| **Security** | ● Per User Server Load Limitations<br>● Enforced TLS Connections |
| **Administration** | ● New functions for User Management<br>● Enhanced Informations from EXPLAIN<br>● User defined variables added to Information Schema<br>● Binary Log based Flashback |
| **Performance** | ● Enhanced Performance for creating Connections<br>● Indexes for Virtual Columns<br>● New Option to define a directory for InnoDB temporary files |
| **Server-Internal Optimisations** | ● Internal Use of MariaDB Connector/C<br>● Optimizer Enhancements<br>● Non-Locking ANALYZE TABLE |
| **Other Enhancements** | ● Lifted limitations for Virtual Computed Columns<br>● Subquery-Support for Views<br>● Multiple Use of Temporary Tables in Query |

# MariaDB Server 10.3 - *Database Compatibility* and *Temporal Data Support* for MariaDB Server

# Overview MariaDB Server 10.3

## Temporal Data processing

- System Versioned Tables store information relating to past and present time.

## Database Compatibility Enhancements

- PL/SQL compatibility parser
- Sequences
- INTERSECT and EXCEPT to complement UNION
- New ROW type  and TYPE OF stored functions
- Invisible Columns

## Performance

- ADD INSTANT COLUMN for InnoDB
- Statement based lock wait timeouts

## Flexibility

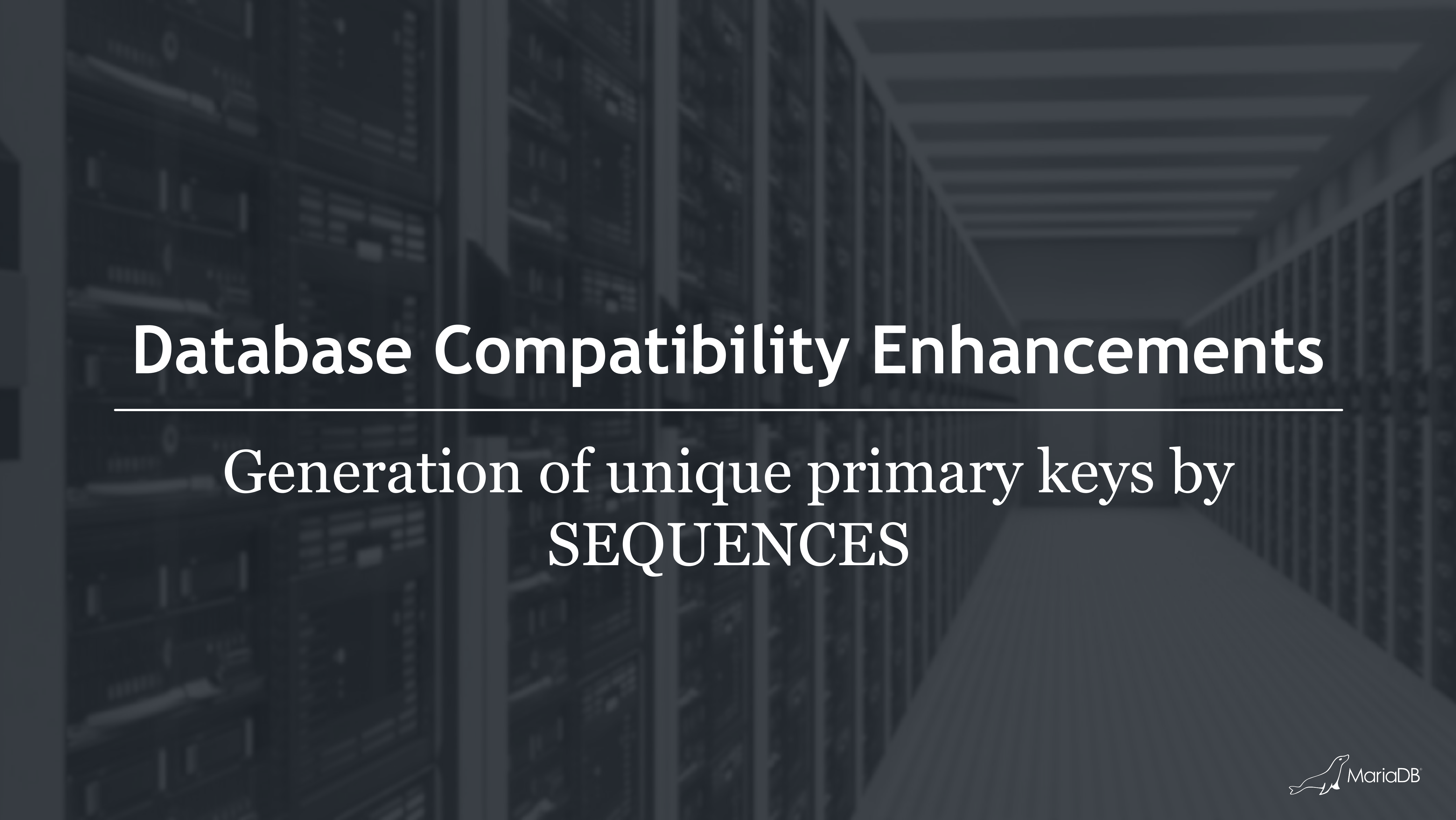
- User defined aggregate functions
- Compressed Columns
- Proxy protocol support

## Scalability

- Spider storage engine updated to the latest release

## Removed limitations

- DELETE statement can delete from the table that is used in a subquery in the WHERE clause
- UPDATE statements can use same source and target



# Database Compatibility Enhancements

## Generation of unique primary keys by SEQUENCES

# Generation of unique primary keys by SEQUENCES

- To enhance the database compatibility with other vendors, MariaDB Server now allows the creation of a SEQUENCE, which is used to create a sequence of numeric values

- Creating a sequence is not replacing the auto increment option, which is well known by MariaDB users, but is an alternative of creating unique identifiers and offers more control of how numbers are created

- Sequences also allow to compute the last number created by all existing sequences, a limitation auto increments have as they only can compute their own last number created.

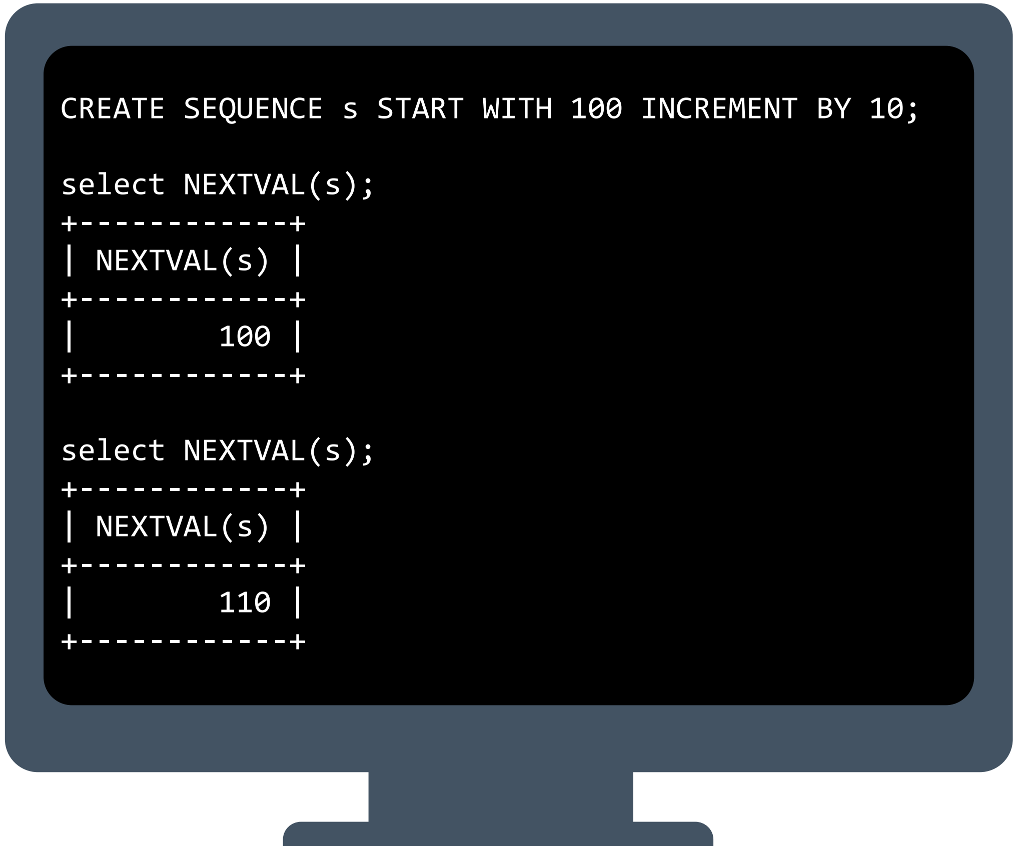
- Sequences are implemented as tables with some exclusive options in MariaDB, which allows that existing tools and processes work unchanged

# Generation of unique primary keys by SEQUENCES Example

- A sequence can be created by the syntax

```
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]
```

- Values of a sequence are returned by
  - `NEXT VALUE FOR sequence_name`
  - `NEXTVAL(sequence_name)`
  - `PREVIOUS VALUE FOR sequence_name`
  - `LASTVAL(sequence_name)`
  - With sql_mode=Oracle are supported
    - `sequence_name.nextval`
    - `sequence_name.currval`



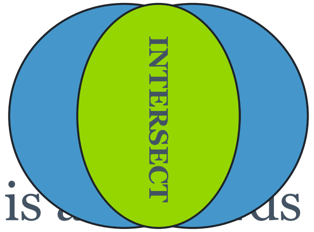
```
CREATE SEQUENCE s START WITH 100 INCREMENT BY 10;

select NEXTVAL(s);
+------------+
| NEXTVAL(s) |
+------------+
|        100 |
+------------+

select NEXTVAL(s);
+------------+
| NEXTVAL(s) |
+------------+
|        110 |
+------------+
```

# Database Compatibility Enhancements

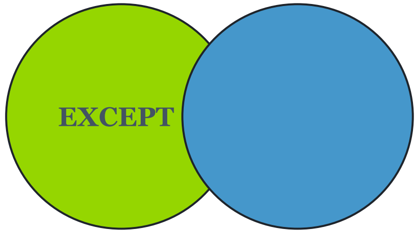
## Operations over result sets with INTERSECT and EXCEPT

# Operations over result sets with INTERSECT and EXCEPT

- MariaDB Server is adding INTERSECT and EXCEPT to UNION, which already exists in former versions
- The result of an INTERSECT is the intersection of right and left SELECT results, i.e. only records that are present in both result sets will be included in the result of the operation.



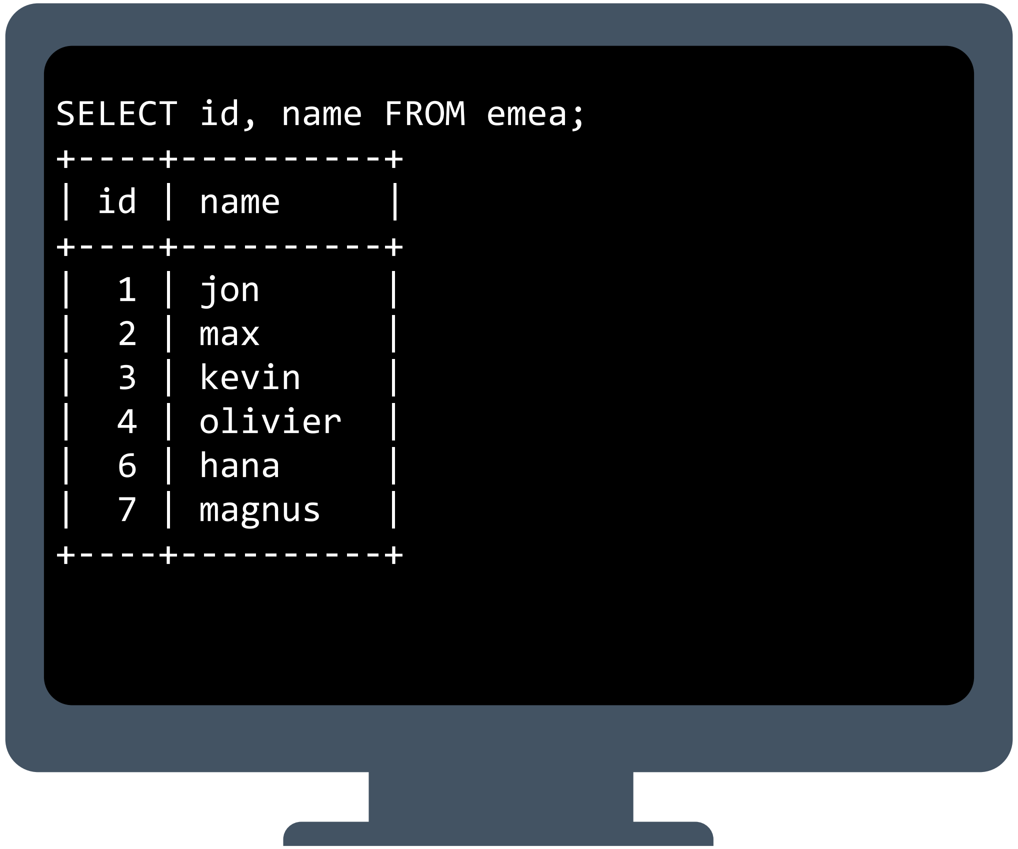
- The result of EXCEPT is a [illegible]ds of the left SELECT result except records which are in right SELECT result set, i.e. it is subtraction of two result sets.



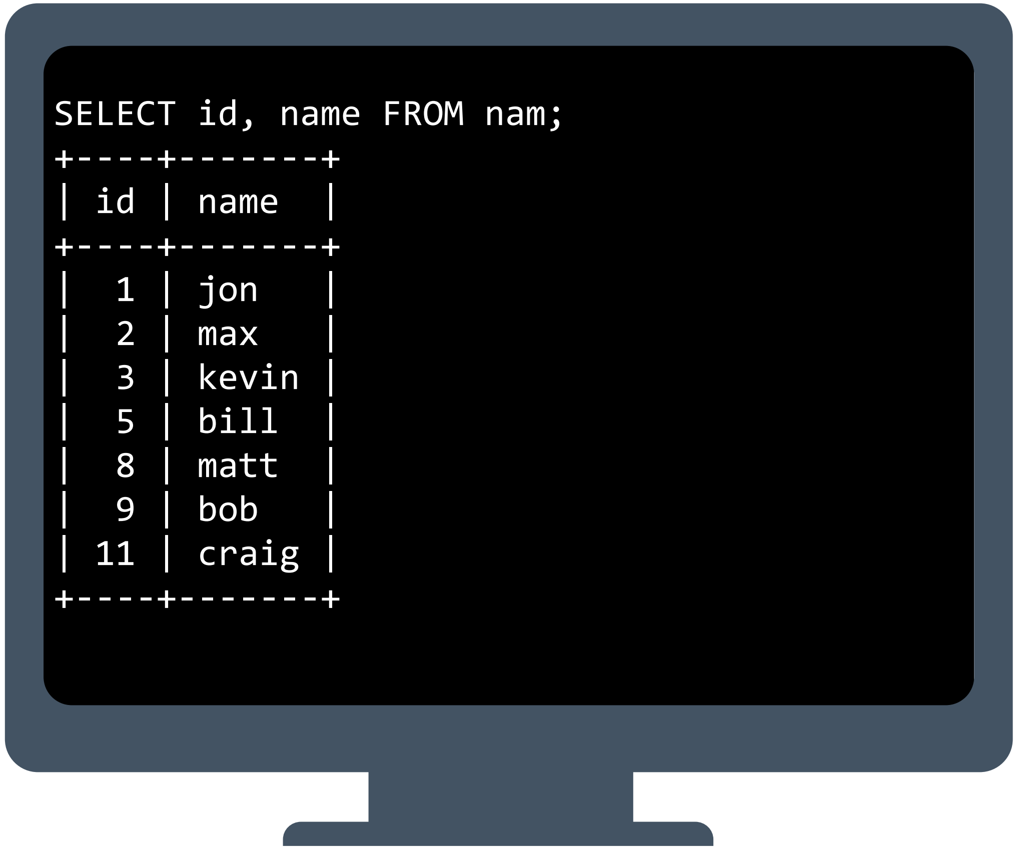
A EXCEPT B

# INTERSECT & EXCEPT

## Example



```
SELECT id, name FROM emea;
+----+----------+
| id | name     |
+----+----------+
|  1 | jon      |
|  2 | max      |
|  3 | kevin    |
|  4 | olivier  |
|  6 | hana     |
|  7 | magnus   |
+----+----------+
```

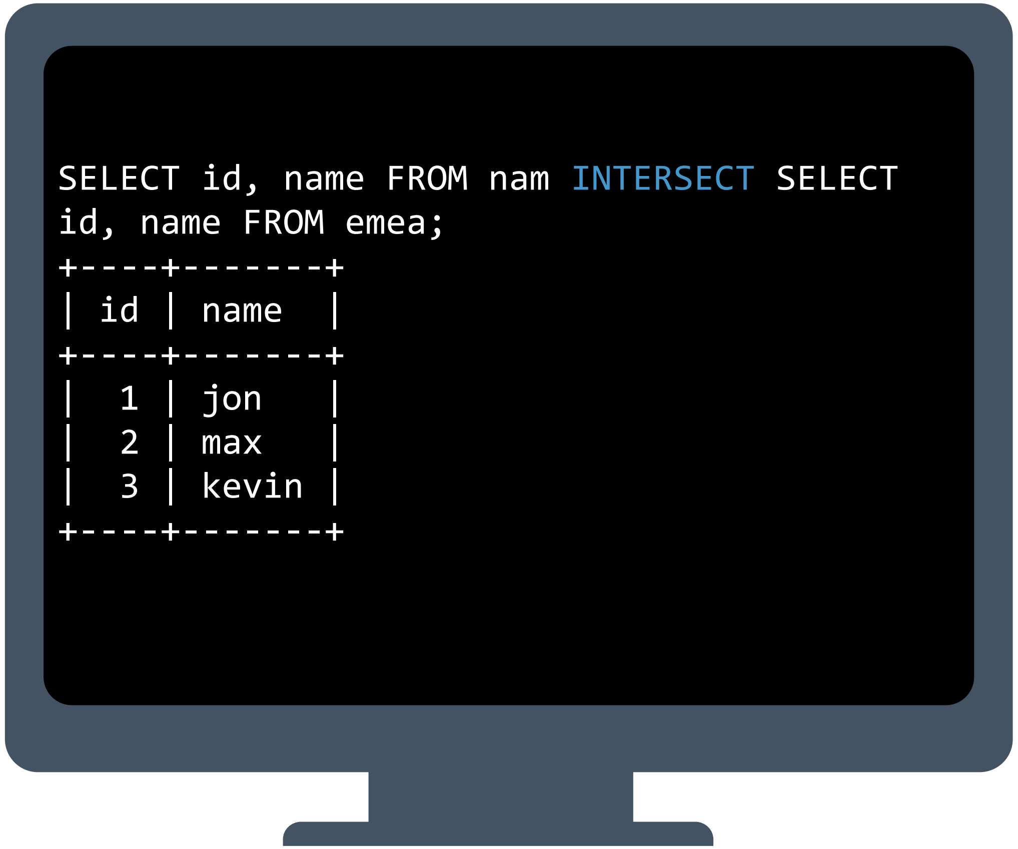


```
SELECT id, name FROM nam;
+----+-------+
| id | name  |
+----+-------+
|  1 | jon   |
|  2 | max   |
|  3 | kevin |
|  5 | bill  |
|  8 | matt  |
|  9 | bob   |
| 11 | craig |
+----+-------+
```

# INTERSECT & EXCEPT

## Example

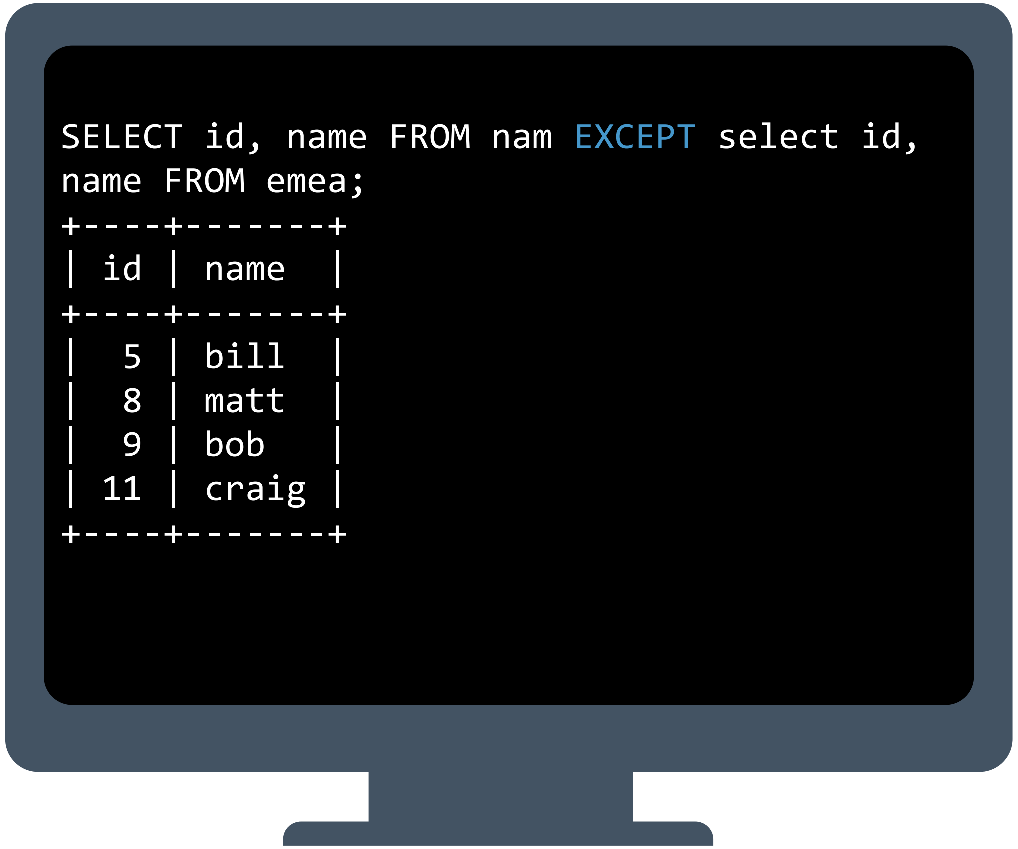
- The names Jon, Max and Kevin are included in both tables, nam and emea



```
SELECT id, name FROM nam INTERSECT SELECT
id, name FROM emea;
+----+-------+
| id | name  |
+----+-------+
|  1 | jon   |
|  2 | max   |
|  3 | kevin |
+----+-------+
```

# INTERSECT & EXCEPT
## Example

- The names Bill, Matt, Bob and Craig exist in table nam, but not in table emea

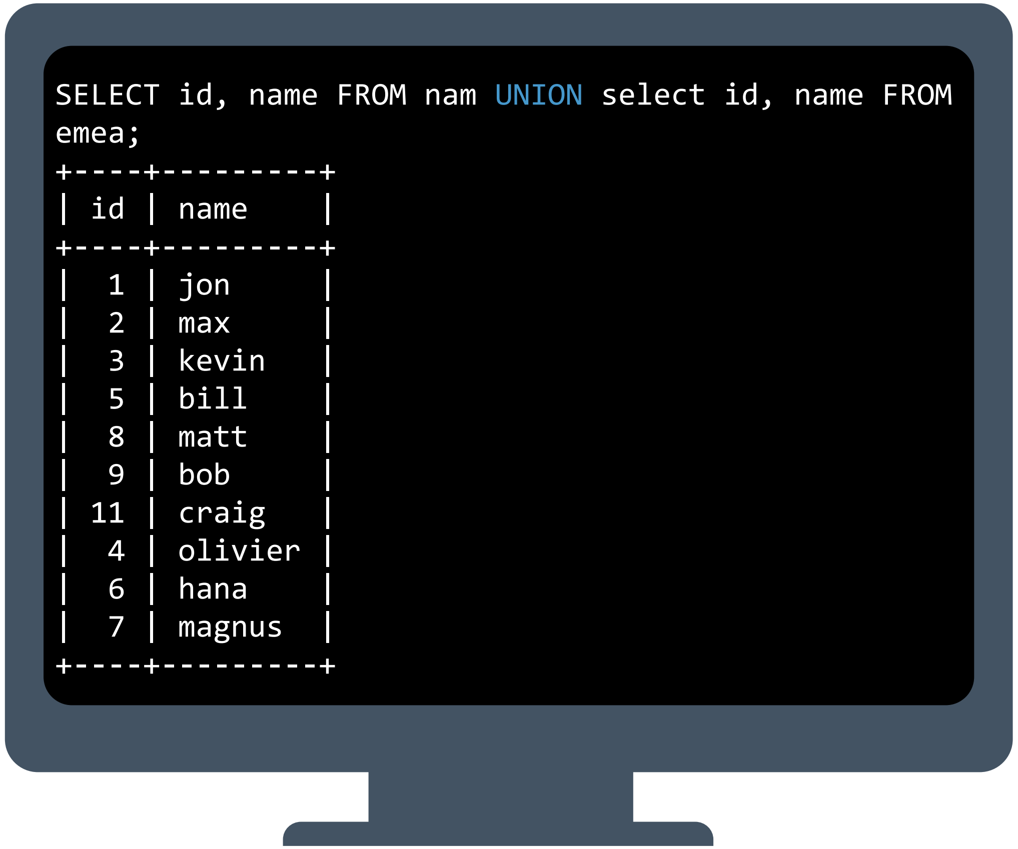


```
SELECT id, name FROM nam EXCEPT select id,
name FROM emea;
+----+-------+
| id | name  |
+----+-------+
|  5 | bill  |
|  8 | matt  |
|  9 | bob   |
| 11 | craig |
+----+-------+
```

# INTERSECT & EXCEPT

## Example

- UNION (existed before 10.3) combines the result from the first and second select



```
SELECT id, name FROM nam UNION select id, name FROM
emea;
+----+---------+
| id | name    |
+----+---------+
|  1 | jon     |
|  2 | max     |
|  3 | kevin   |
|  5 | bill    |
|  8 | matt    |
|  9 | bob     |
| 11 | craig   |
|  4 | olivier |
|  6 | hana    |
|  7 | magnus  |
+----+---------+
```

# Database Compatibility Enhancements

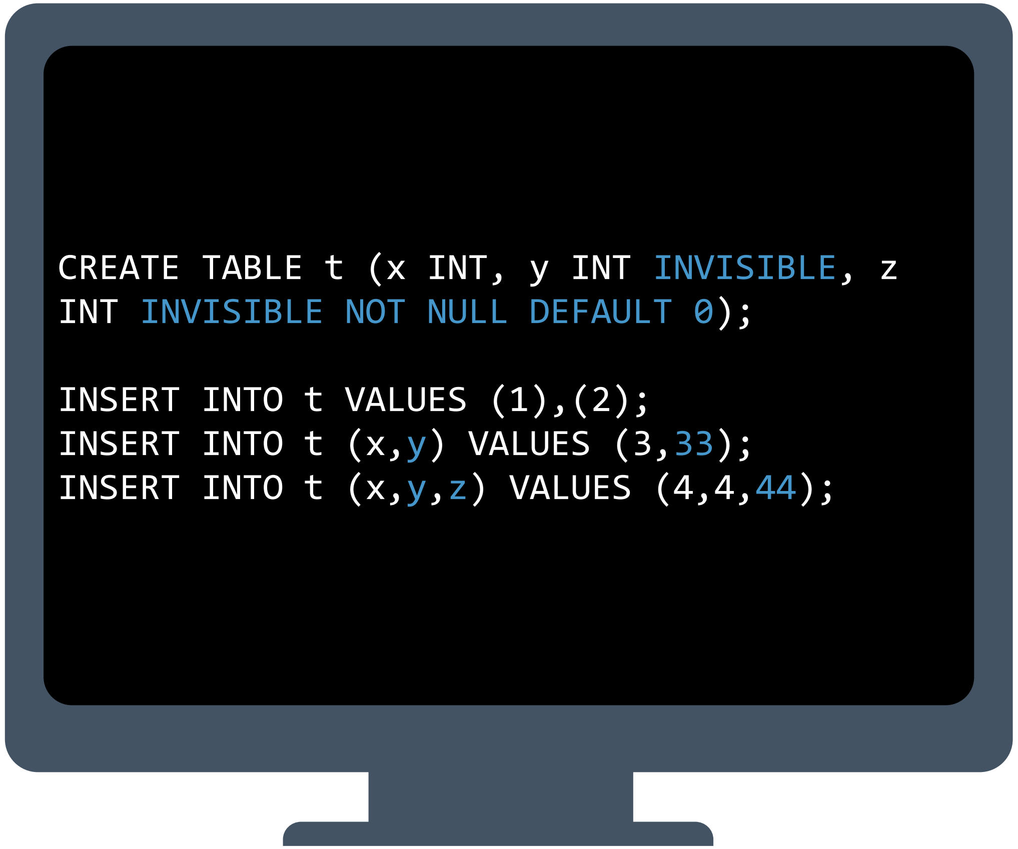
## Invisible Columns

# Define Columns to be invisible if not explicitly queried

- Invisible columns allow to remove dependency to applications
  - columns can be added to tables with hiding them from the application, which otherwise might fail to run
  - keep historical columns not needed by applications anymore
  - prepare the database level for an upgrade before an application will be upgraded
  - hide system created columns
- An enhanced syntax allows to define a column as “invisible”
  - “select * from table” will not show columns, which are defined invisible
  - An explicit mentioning will show the column in a result set
- Creating a table with a column defined as invisible can be NOT NULL, when DEFAULT is given
- An INSERT statement does not require a value to be provided for a invisible column

# Invisible Columns
# Example

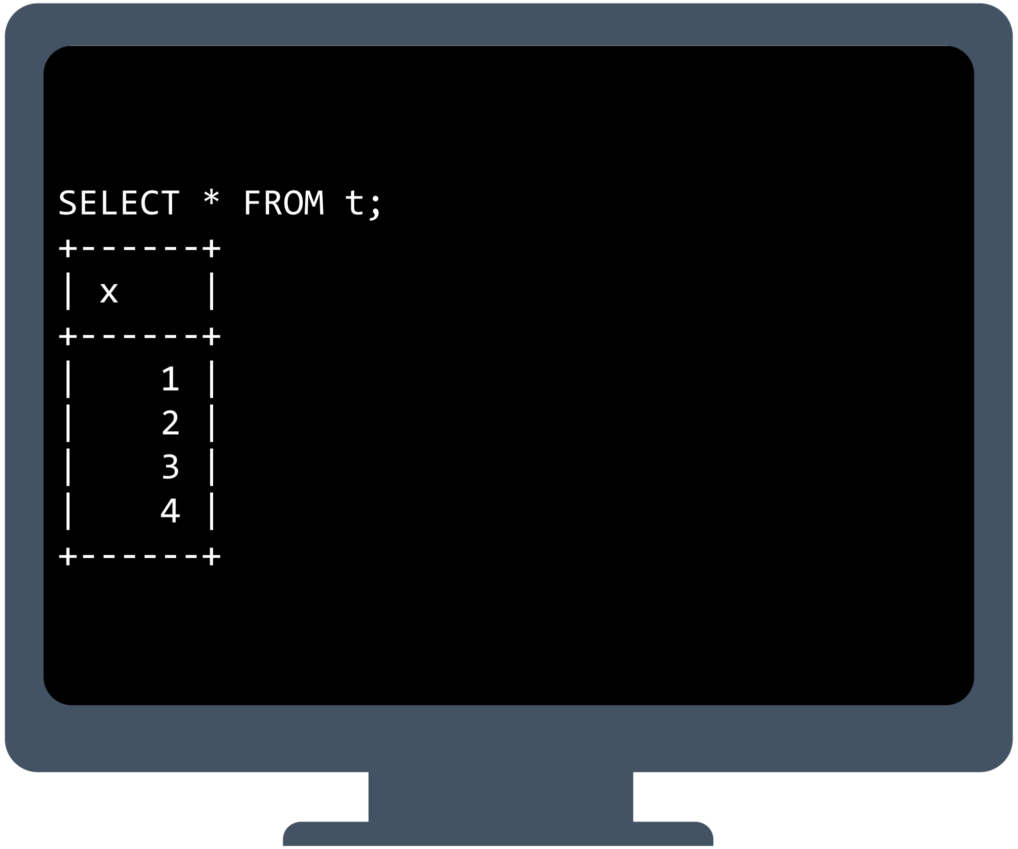
- Create a table t with invisible columns y and z
    - If NOT NULL is used a DEFAULT value has to be defined
- For INSERT the invisible fields need to be specified, if data will be added



```
CREATE TABLE t (x INT, y INT INVISIBLE, z
INT INVISIBLE NOT NULL DEFAULT 0);

INSERT INTO t VALUES (1),(2);
INSERT INTO t (x,y) VALUES (3,33);
INSERT INTO t (x,y,z) VALUES (4,4,44);
```

# Invisible Columns
# Example

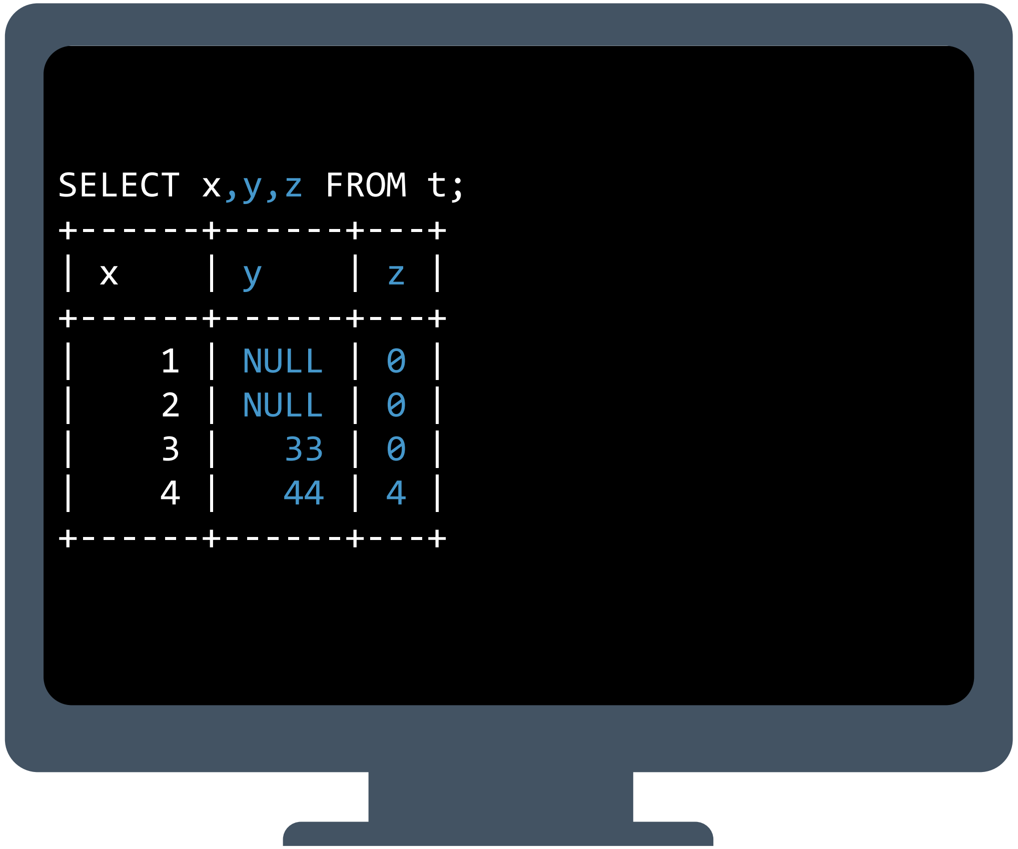
- Selecting from the table without specifying the fields only shows visible fields



```
SELECT * FROM t;
+-------+
| x     |
+-------+
|     1 |
|     2 |
|     3 |
|     4 |
+-------+
```

# Invisible Columns
# Example

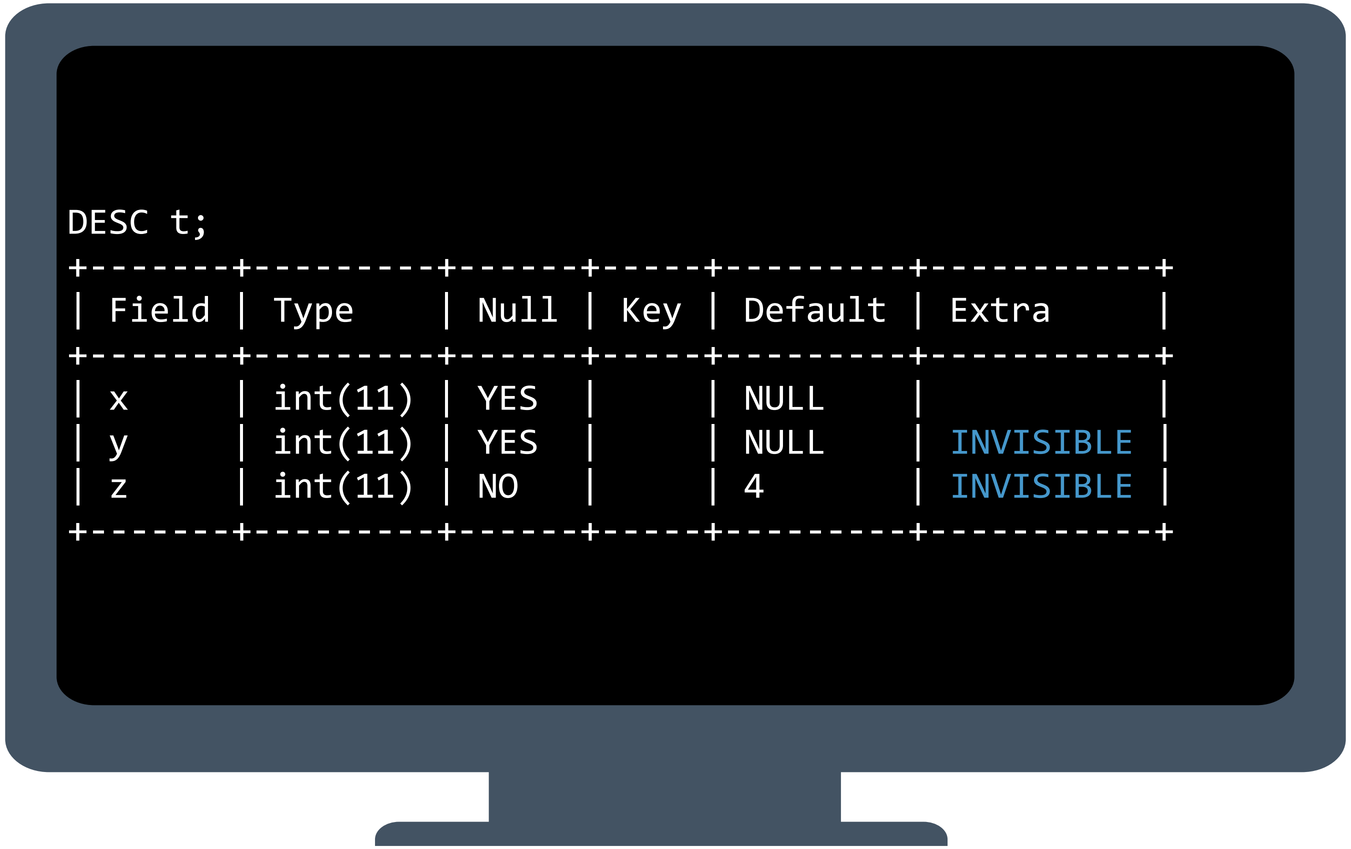
- Selecting from the table with specifying the fields y and z also shows invisible fields



```
SELECT x,y,z FROM t;
+------+------+---+
| x    | y    | z |
+------+------+---+
|    1 | NULL | 0 |
|    2 | NULL | 0 |
|    3 |   33 | 0 |
|    4 |   44 | 4 |
+------+------+---+
```

# Invisible Columns Example

- Invisible columns are marked in DESCRIBE in the field “Extra”



```
DESC t;
+-------+---------+------+-----+---------+-----------+
| Field | Type    | Null | Key | Default | Extra     |
+-------+---------+------+-----+---------+-----------+
| x     | int(11) | YES  |     | NULL    |           |
| y     | int(11) | YES  |     | NULL    | INVISIBLE |
| z     | int(11) | NO   |     | 4       | INVISIBLE |
+-------+---------+------+-----+---------+-----------+
```

# Database Compatibility Enhancements

## PL/SQL Compatibility for MariaDB Stored Functions

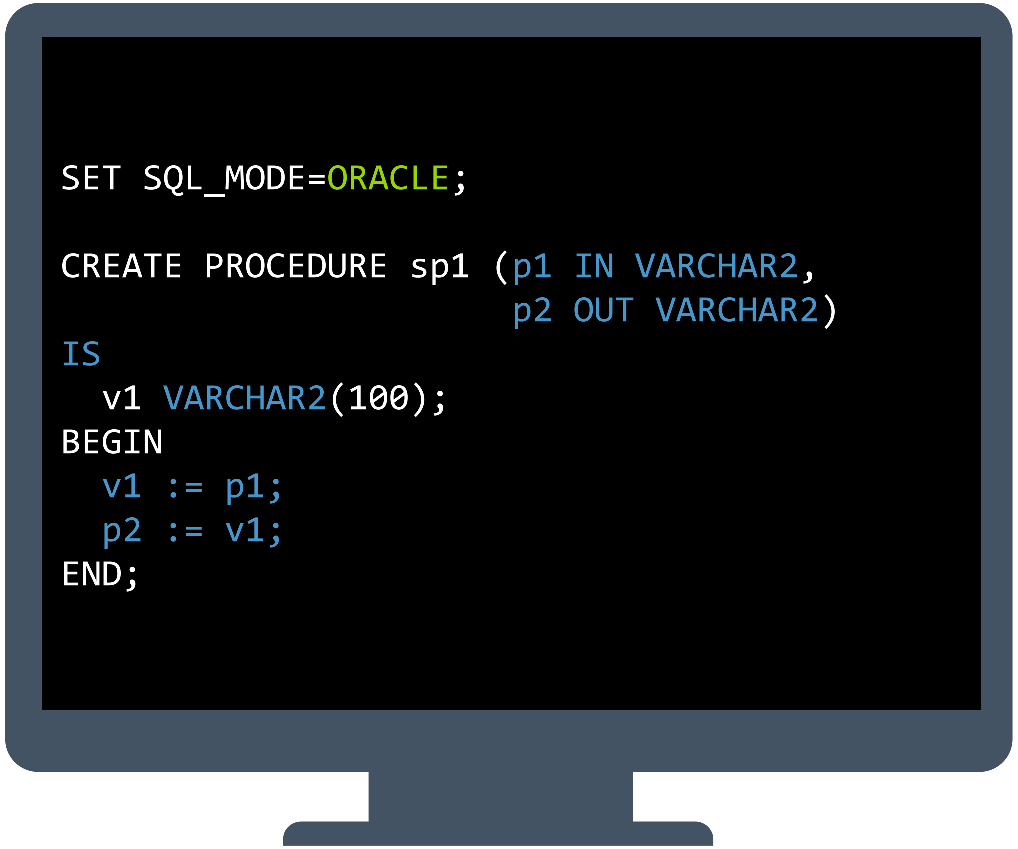
# PL/SQL Compatibility

for MariaDB Stored Functions including packages

- PL/SQL compatibility parser added for easier migration from Oracle to MariaDB
  - No need to migrate Oracle PL/SQL logic to SQL/PSM or to the application layer, when migrating to MariaDB
- Compatibility “Oracle SQL Mode” is used when syntax is not compatible to SQL/PSM standard
  - sql_mode=’oracle’
  - Existing SQL/PSM based Stored Functions can still be used
- Syntactic differences between SQL/PSM and PL/SQL are addressed by the compatibility parser

# Compatibility parser

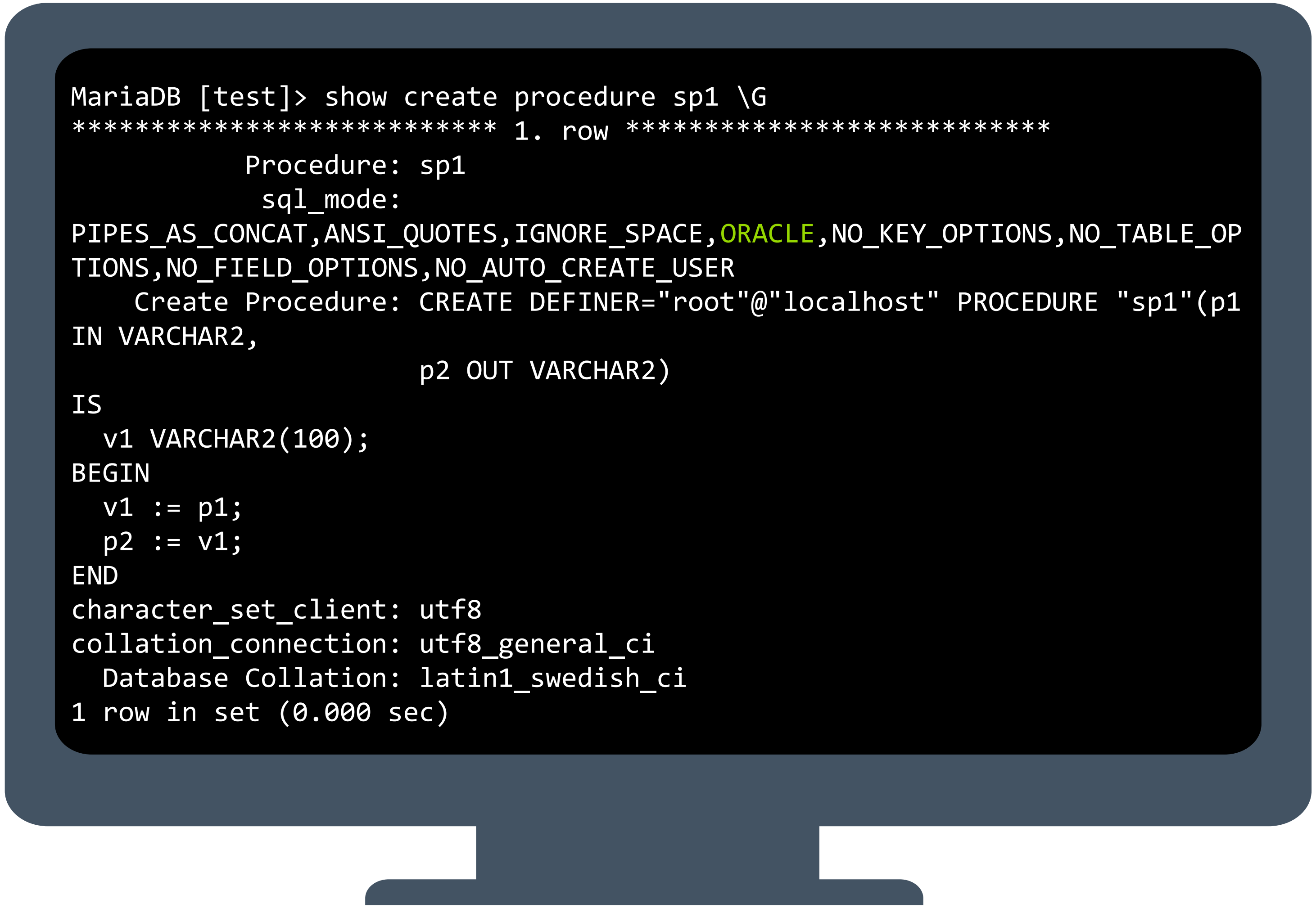
- Set SQL_MODE to Oracle when creating a Stored Procedure using the Oracle PL/Syntax
  - Understands a large subset of Oracle's PL/SQL language instead of MariaDB's traditional syntax for stored routines
- Syntax Error if wrong SQL mode was set



```
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

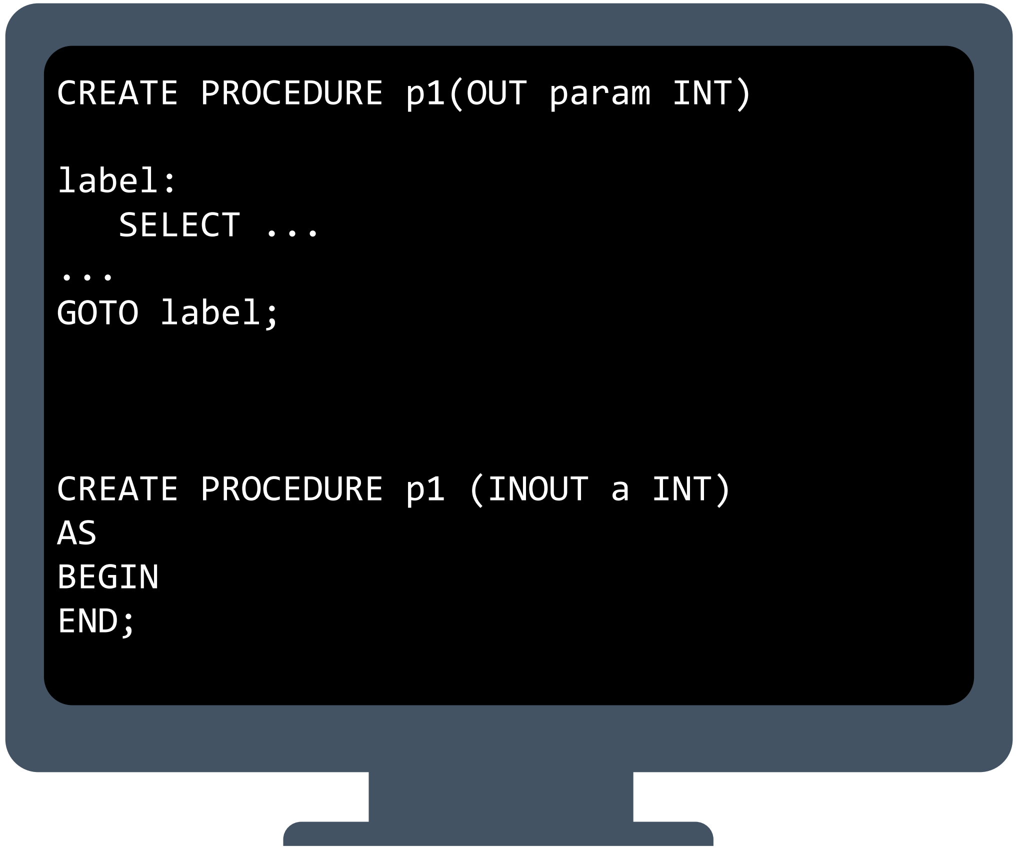
- The SQL mode is stored for each stored procedure in information_schema.routines
  - MariaDB and Oracle syntax based stored procedures can be used in the same database



```
MariaDB [test]> show create procedure sp1 \G
*************************** 1. row ***************************
           Procedure: sp1
            sql_mode:
PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ORACLE,NO_KEY_OPTIONS,NO_TABLE_OP
TIONS,NO_FIELD_OPTIONS,NO_AUTO_CREATE_USER
    Create Procedure: CREATE DEFINER="root"@"localhost" PROCEDURE "sp1"(p1
IN VARCHAR2,
                      p2 OUT VARCHAR2)
IS
  v1 VARCHAR2(100);
BEGIN
  v1 := p1;
  p2 := v1;
END
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.000 sec)
```

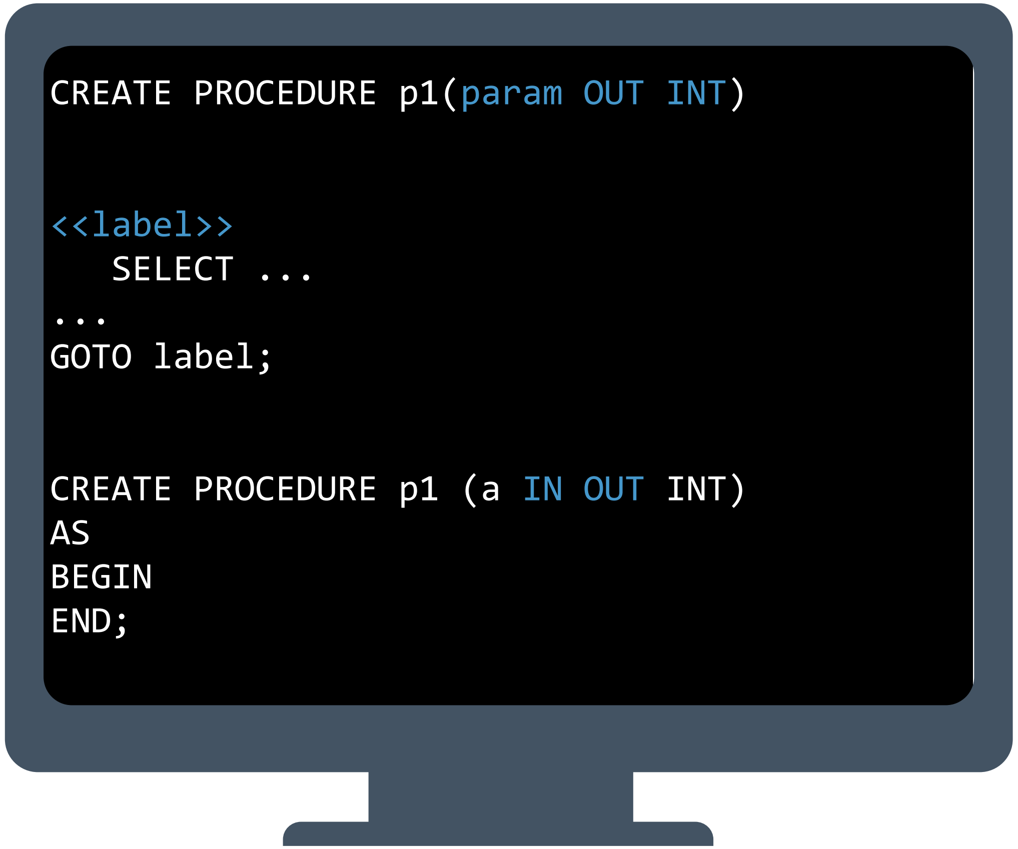
# Compatibility parser
# Label and IN, OUT, INOUT

## MariaDB Syntax



```
CREATE PROCEDURE p1(OUT param INT)

label:
   SELECT ...
...
GOTO label;



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

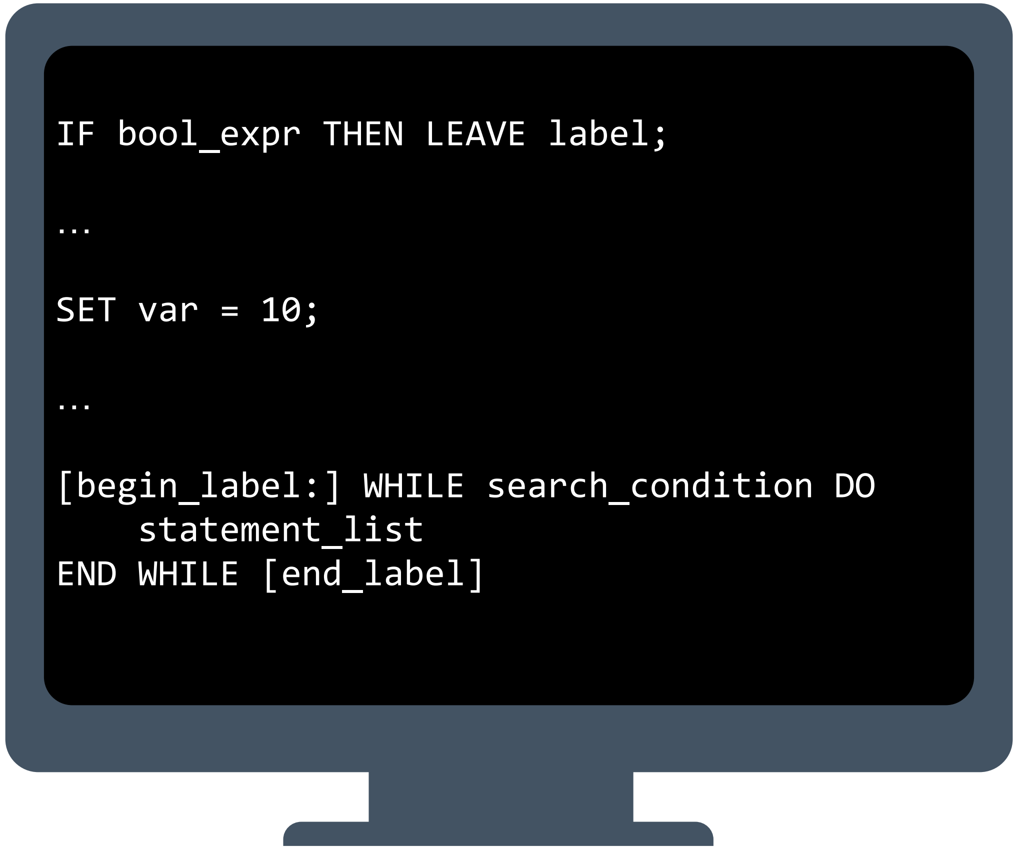
## Oracle Syntax



```
CREATE PROCEDURE p1(param OUT INT)


<<label>>
   SELECT ...
...
GOTO label;


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

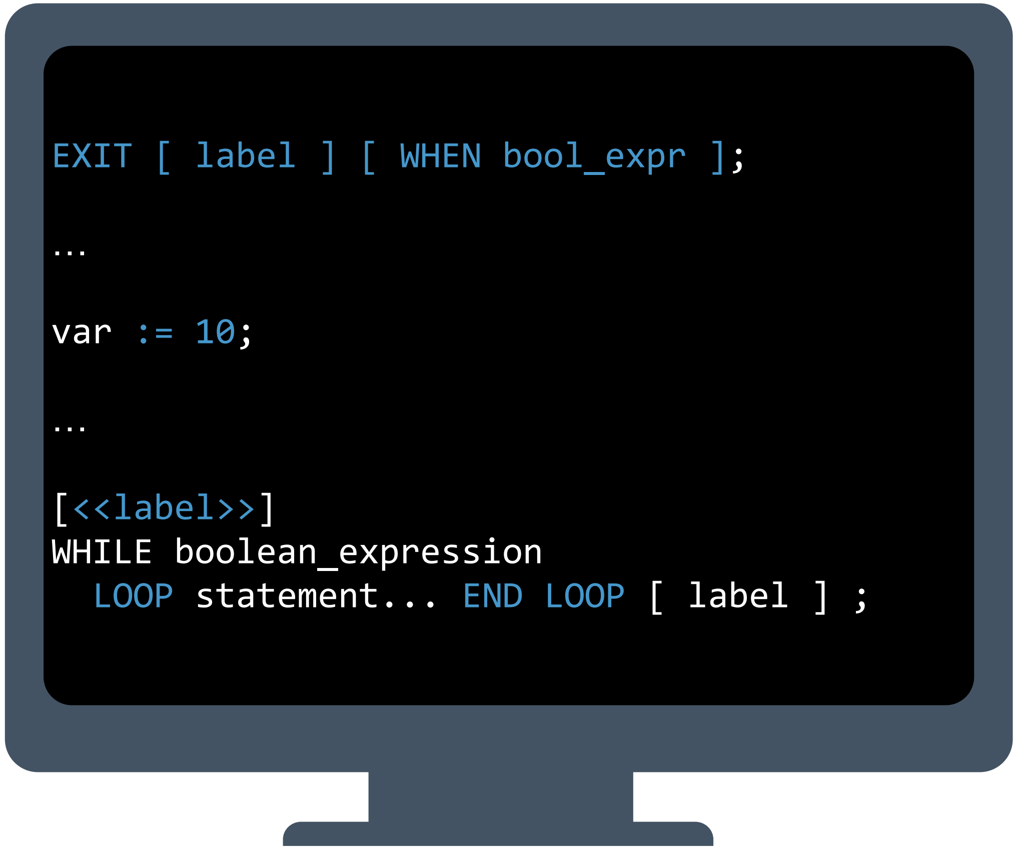
# Compatibility parser
# Other Examples

## MariaDB Syntax



```
IF bool_expr THEN LEAVE label;

…

SET var = 10;

…

[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]
```

## Oracle Syntax

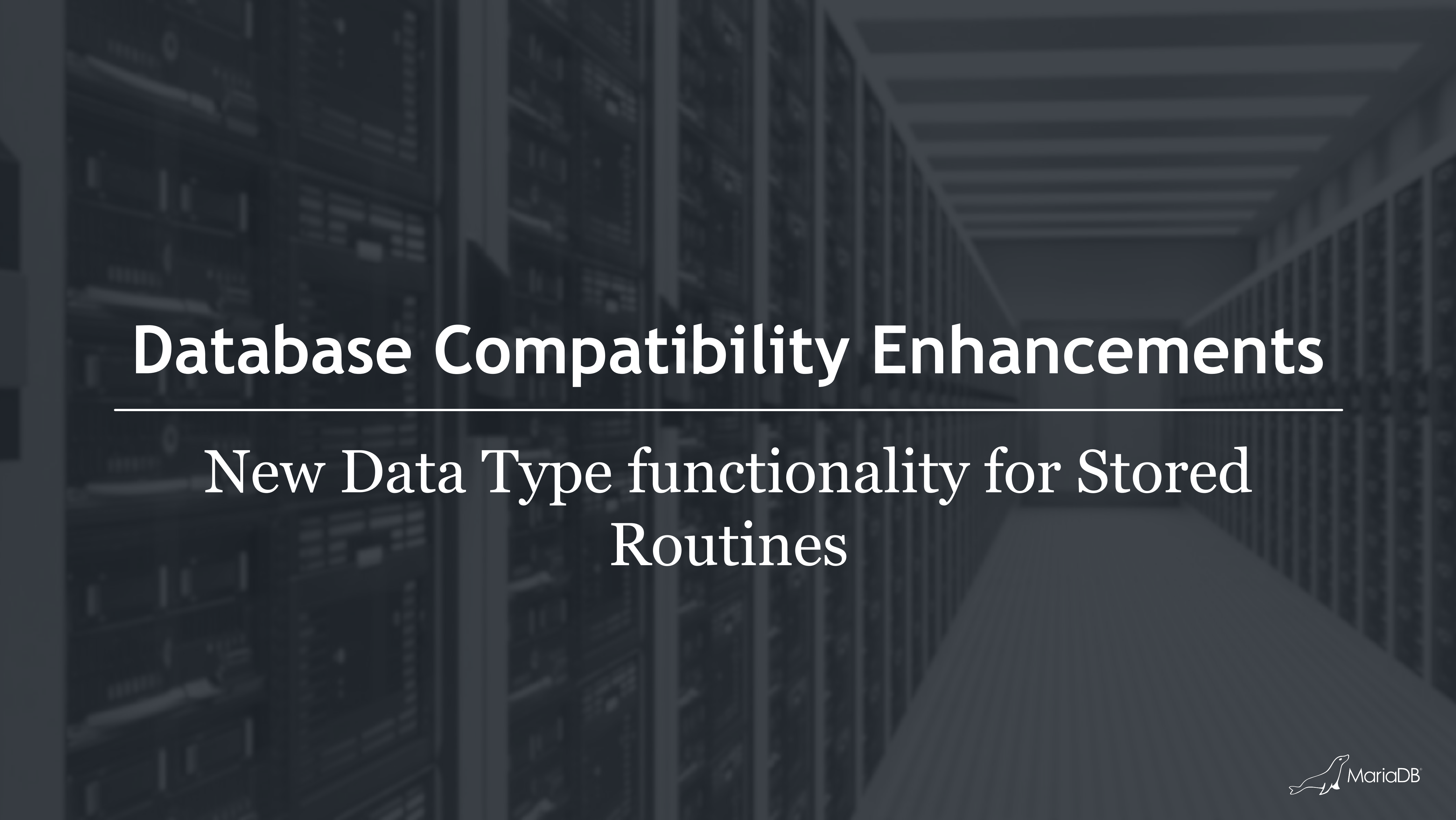


```
EXIT [ label ] [ WHEN bool_expr ];

…

var := 10;

…

[<<label>>]
WHILE boolean_expression
  LOOP statement... END LOOP [ label ] ;
```

MariaDB

# Compatibility parser 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



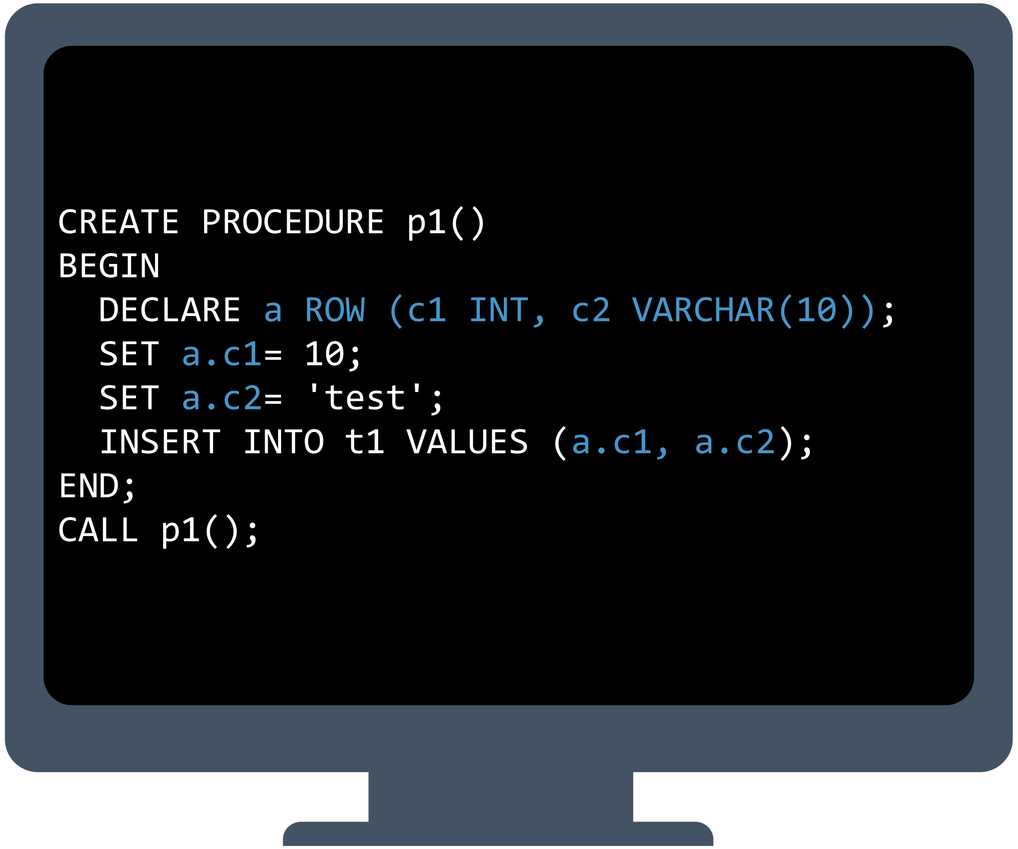
# Database Compatibility Enhancements

## New Data Type functionality for Stored Routines

# Database Compatibility
# ROW data type for stored routines

- A variable can be defined as of type ROW
  - like creating a database table table
  - as working with an array

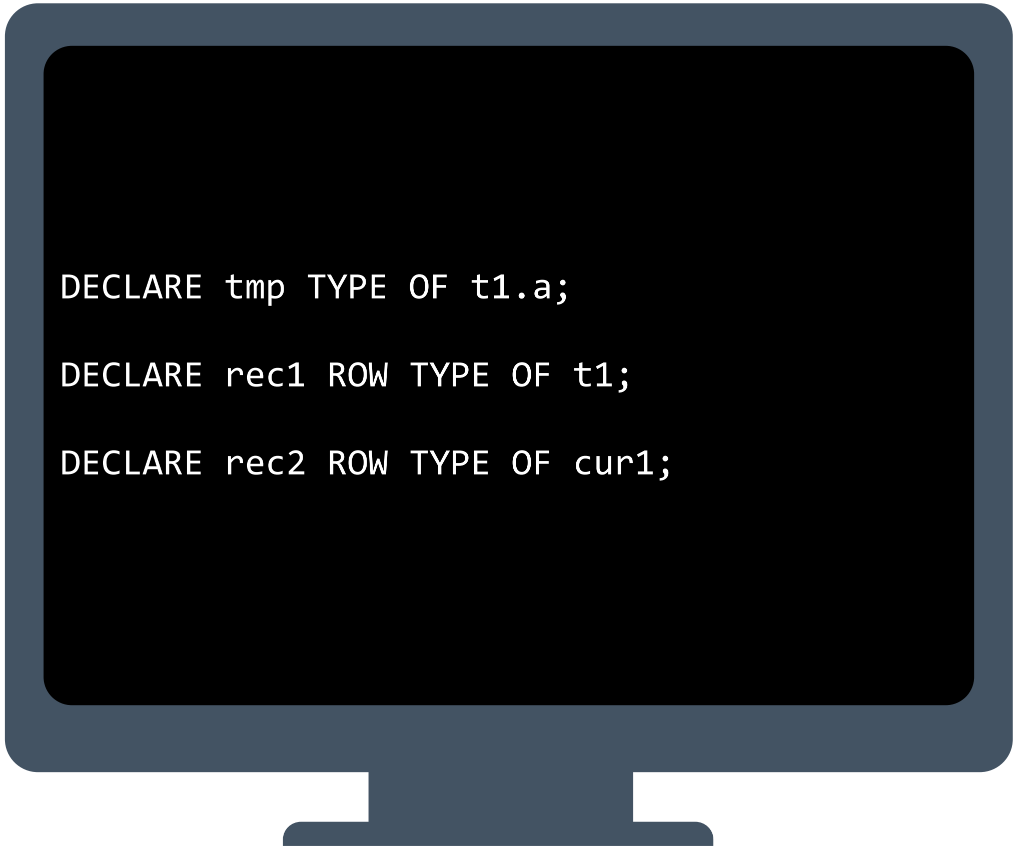
```
ROW data type
<row type>         ::= ROW <row type body>
<row type body>    ::= <left paren> <field definition>
                       [ { <comma> <field definition>
                                     }... ] <right paren>
<field definition> ::= <field name> <data type>
<data type>        ::= <predefined 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();
```

# Database Compatibility
# TYPE OF and ROW TYPE OF for stored routines

- A variable can be declared to inherit its type from an existing type or row type
  - get the data TYPE OF a column in a table
  - get the ROW data TYPE OF a table
  - get the ROW data TYPE OF a cursor

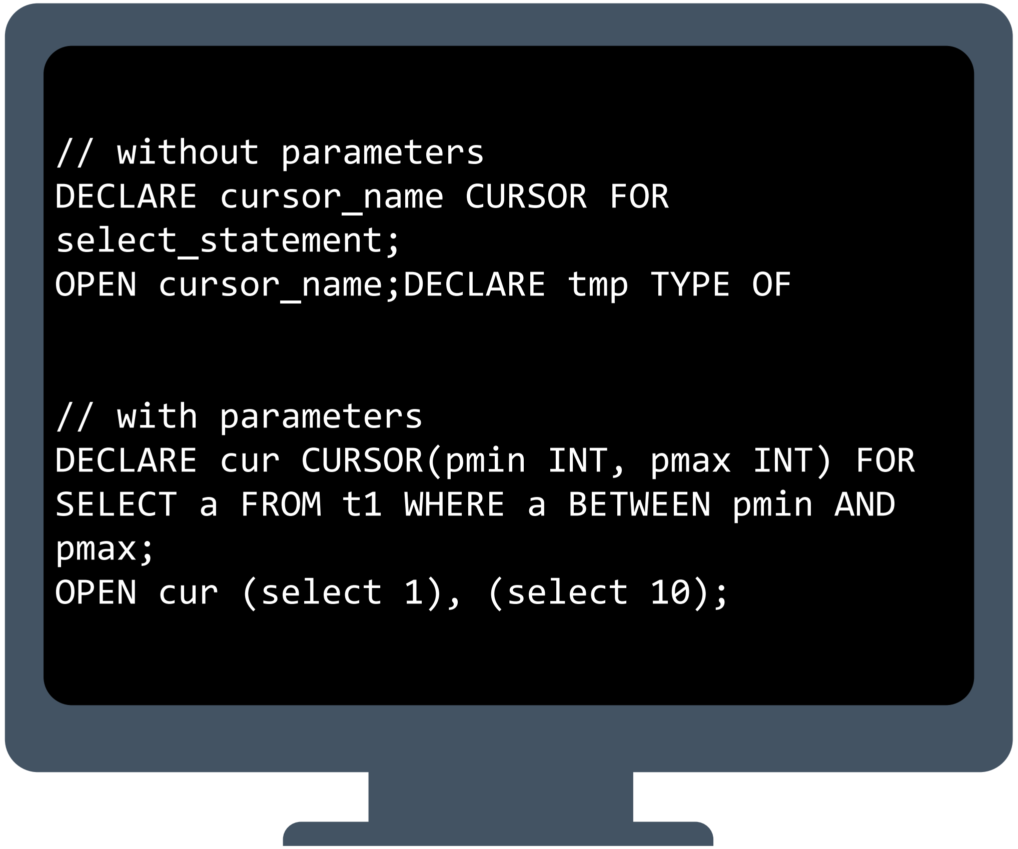


```
DECLARE tmp TYPE OF t1.a;

DECLARE rec1 ROW TYPE OF t1;

DECLARE rec2 ROW TYPE OF cur1;
```

# Database Compatibility
# Cursors with Parameters

- With MariaDB Server 10.3 it is now possible to declare cursors with parameters

```
DECLARE cursor_name [cursor_formal_parameter [, ...]]
CURSOR FOR select_statement;

  <cursor_formal_parameter>::=name type [collate clause]

OPEN cursor_name [expression [, ...]];
```



```
// without parameters
DECLARE cursor_name CURSOR FOR
select_statement;
OPEN cursor_name;DECLARE tmp TYPE OF


// with parameters
DECLARE cur CURSOR(pmin INT, pmax INT) FOR
SELECT a FROM t1 WHERE a BETWEEN pmin AND
pmax;
OPEN cur (select 1), (select 10);
```

# Analytics / Temporal data processing

System Versioned Tables with AS OF Query enhancement

# Temporal Data processing

System Versioned Tables with AS OF Query enhancement

- SQL support for time-related information
  - The database can store all versions of stored records
- A table can be altered to enable, disable or remove system versioned data - (transparent to existing applications)
- System versioned tables can be queried using
  - `AS OF` to select data "as of" a given point in time
  - BETWEEN .. AND to select data which has been visible in between two point in times
  - `ALL` to show current and all historical versions
- A new partitioning `BY SYSTEM_TIME` exists to partition data separately for
  - historical data
  - currently valid data
- Historical data can be removed from a System versioned table by a new syntax `DELETE HISTORY`

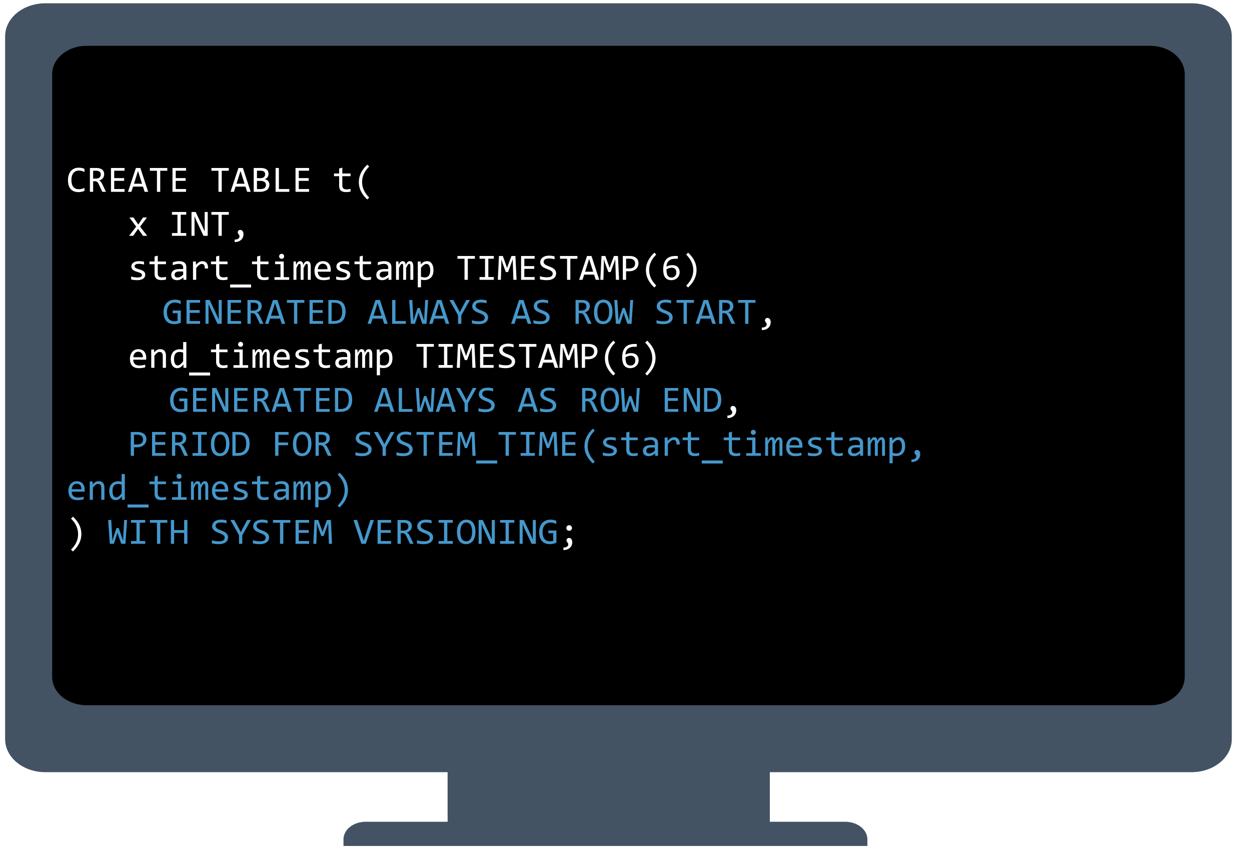
# Temporal Data processing

System Versioned Tables with AS OF Query enhancement

- System Versioned Tables are used for
  - Data analysis (retrospective, trends etc.)
  - Forensic discovery & legal requirements to store data for N years (data auditing, OLTP with data history )
  - Point-in-time recovery
- System Versioned Tables include timestamped versions of the data in a table. This allows
  - to track changes
  - to compare data based on timestamps
  - to visualize the development cycle of data and to create trends
  - to audit the change of data

# System Version Tables
# Creating the Table

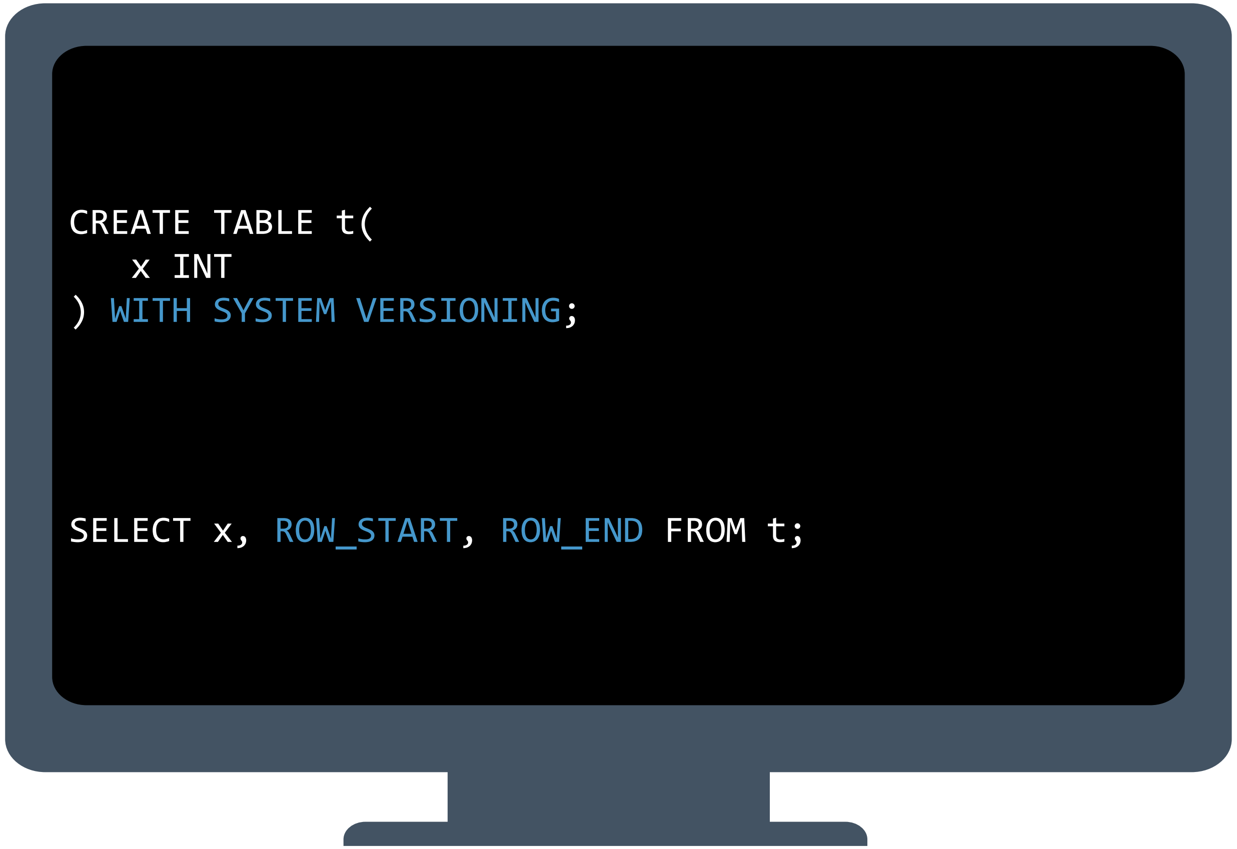
- A System versioned table, according to the SQL:2011, have
  - two generated columns to define the start and end time the data is valid for
  - a period definition
  - the special table option clause WITH SYSTEM VERSIONING



```
CREATE TABLE t(
   x INT,
   start_timestamp TIMESTAMP(6)
     GENERATED ALWAYS AS ROW START,
   end_timestamp TIMESTAMP(6)
     GENERATED ALWAYS AS ROW END,
   PERIOD FOR SYSTEM_TIME(start_timestamp,
end_timestamp)
) WITH SYSTEM VERSIONING;
```

# System Version Tables
# Creating the Table

- MariaDB allows a simplified syntax, when only the table option clause WITH SYSTEM VERSIONING is used
  - the two generated columns are generated as invisible fields `row_start` and `row_end`
  - a period definition defined but invisible
- To query the versioning fields `ROW_START` and `ROW_END`, they have to be listed in the SELECT



```
CREATE TABLE t(
   x INT
) WITH SYSTEM VERSIONING;



SELECT x, ROW_START, ROW_END FROM t;
```

# AS OF Example

```
create table emp (
    name varchar(30), salary int, dept int
) with system versioning
```

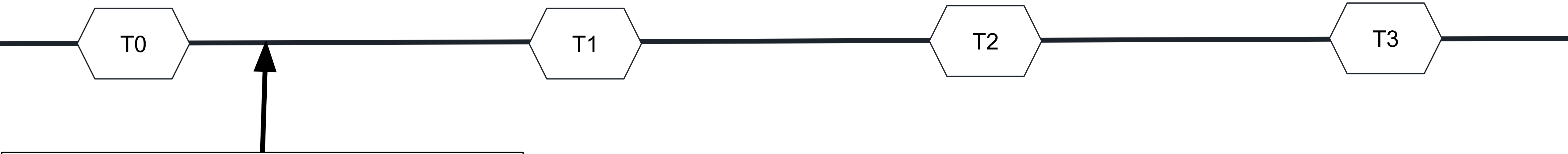
| name | salary | dept |
|---|---|---|

# AS OF Example

```
create table emp (
    name varchar(30), salary int, dept int
) with system versioning
```

| name | salary | dept |
|---|---|---|

| name | salary | dept |
|---|---|---|
| Bill | 1000 | 10 |



```
insert into emp
values (“Bill”, 1000, 10)
```

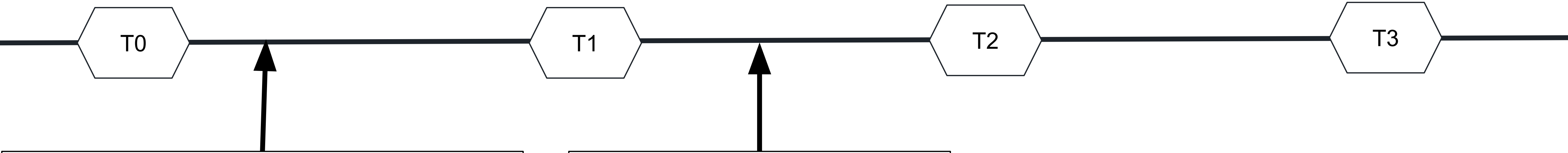
# AS OF Example

```
create table emp (
    name varchar(30), salary int, dept int
) with system versioning
```

| name | salary | dept |
|---|---|---|

| name | salary | dept |
|---|---|---|
| Bill | 1000 | 10 |

| name | salary | dept |
|---|---|---|
| Bill | 2000 | 10 |



```
insert into emp
values (“Bill”, 1000, 10)
```

```
update emp
set salary=2000
where name = “Bill”
```

# AS OF Example

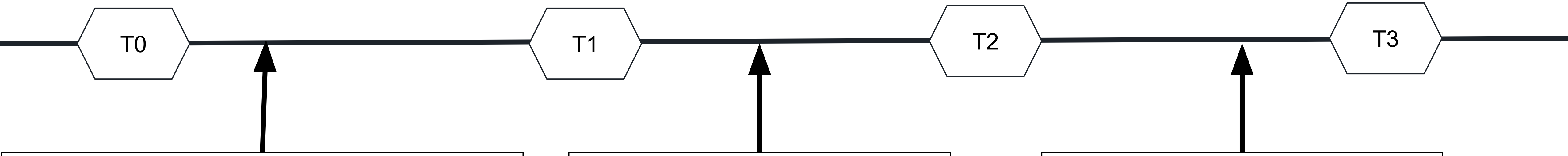
```
create table emp (
    name varchar(30), salary int, dept int
) with system versioning
```

| name | salary | dept |
|---|---|---|

| name | salary | dept |
|---|---|---|
| Bill | 1000 | 10 |

| name | salary | dept |
|---|---|---|
| Bill | 2000 | 10 |

| name | salary | dept |
|---|---|---|
| Bill | 2000 | 20 |



```
insert into emp
values (“Bill”, 1000, 10)
```

```
update emp
set salary=2000
where name = “Bill”
```

```
update emp
set dept=20
where name = “Bill”
```

# AS OF Example

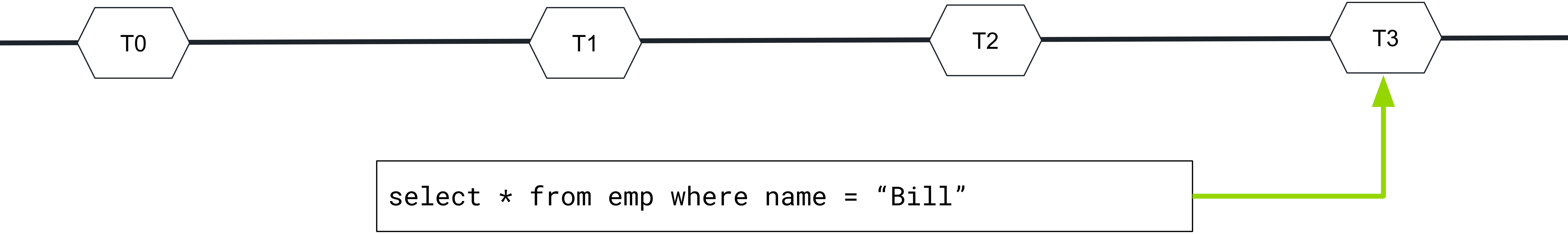
```
create table emp (
    name varchar(30), salary int, dept int
) with system versioning
```

| name | salary | dept |
|---|---|---|

| name | salary | dept |
|---|---|---|
| Bill | 1000 | 10 |

| name | salary | dept |
|---|---|---|
| Bill | 2000 | 10 |

| name | salary | dept |
|---|---|---|
| Bill | 2000 | 20 |



```
select * from emp where name = “Bill”
```

# AS OF Example

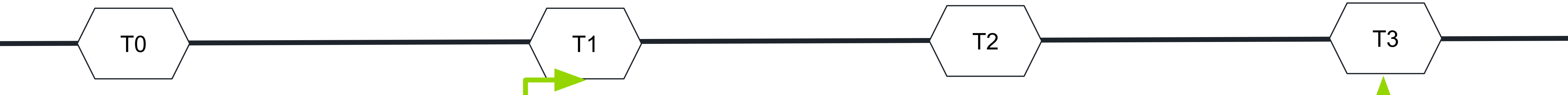
```
create table emp (
    name varchar(30), salary int, dept int
) with system versioning
```

| name | salary | dept |
|---|---|---|

| name | salary | dept |
|---|---|---|
| Bill | 1000 | 10 |

| name | salary | dept |
|---|---|---|
| Bill | 2000 | 10 |

| name | salary | dept |
|---|---|---|
| Bill | 2000 | 20 |



```
select * from emp where name = “Bill”
```

```
select *
from emp for system_time as of timestamp @t1
where name = “Bill”
```

# AS OF Example

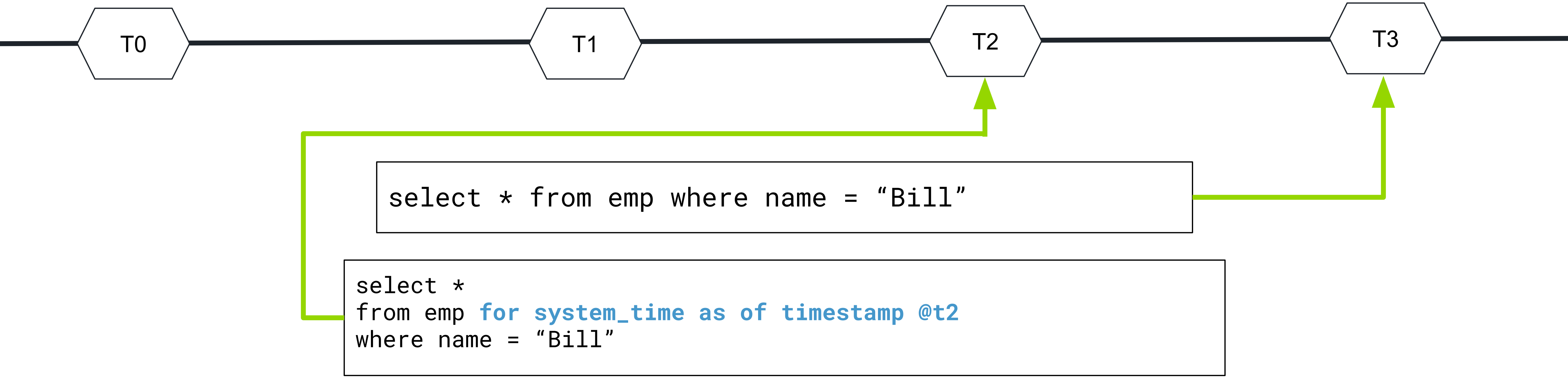
```
create table emp (
    name varchar(30), salary int, dept int
) with system versioning
```

| name | salary | dept |
|---|---|---|

| name | salary | dept |
|---|---|---|
| Bill | 1000 | 10 |

| name | salary | dept |
|---|---|---|
| Bill | 2000 | 10 |

| name | salary | dept |
|---|---|---|
| Bill | 2000 | 20 |



```
select * from emp where name = “Bill”
```

```
select *
from emp for system_time as of timestamp @t2
where name = “Bill”
```

# User Flexibility

## User Defined Aggregate Functions

# User Defined Aggregate Functions

- Aggregate functions are functions where the values of multiple rows are grouped together to form a single value of more significant meaning or measurement
- Aggregate functions are used with the GROUPED BY clause and in Window functions
- MariaDB Server already provides common aggregate functions like avg(), count(), max(), min(), std(), sum()
- User defined aggregate functions allows to create SQL based functions for aggregations
  - new CREATE AGGREGATE FUNCTION
  - FETCH GROUP NEXT ROW as the essential instruction for the aggregate

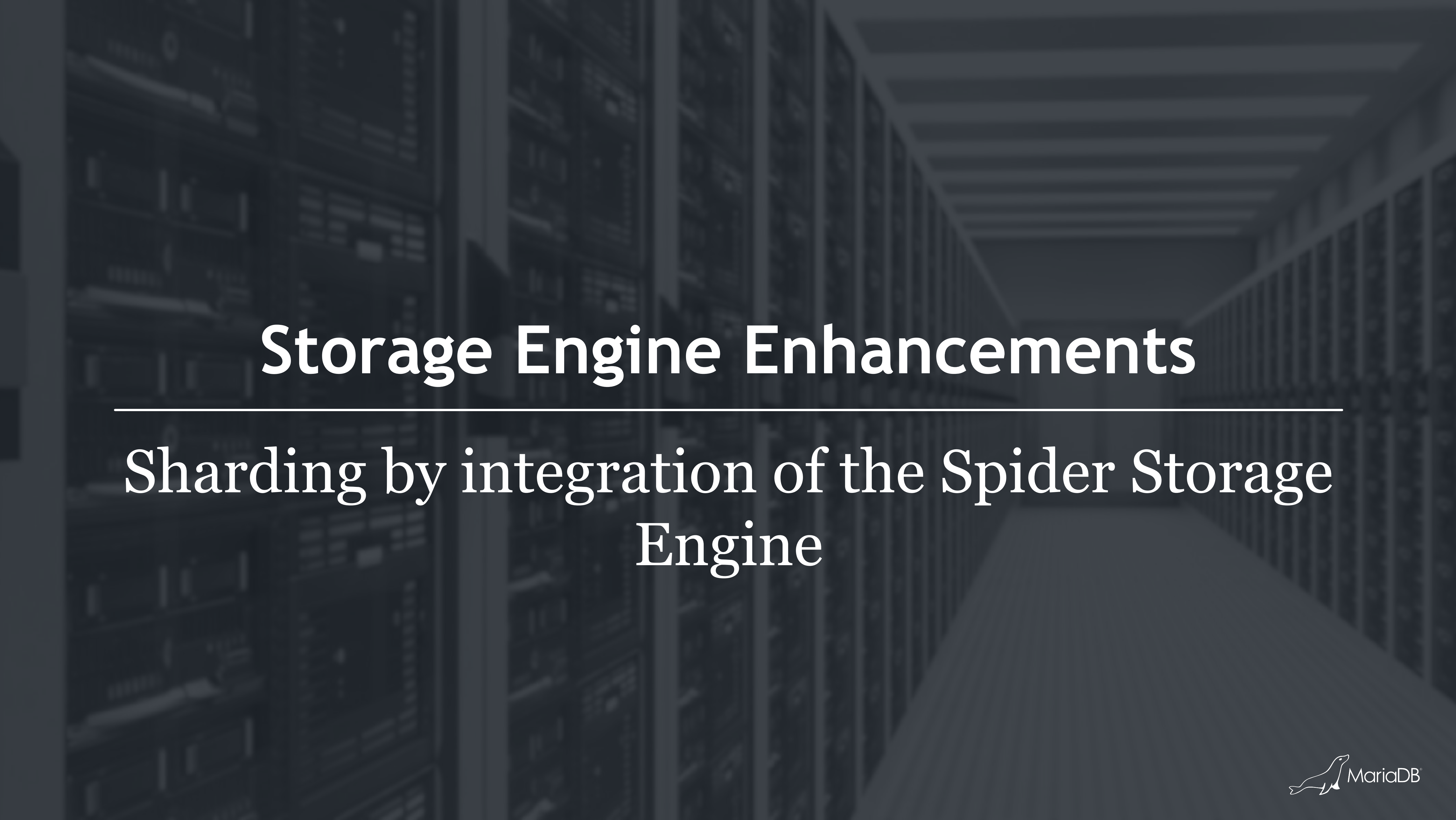
# Custom Aggregate Functions

- Define functions that can be used for aggregation like SUM or AVG

```
CREATE AGGREGATE FUNCTION function_name (param, [param])
RETURNS return_type
  BEGIN
     [variable declarations]
     DECLARE CONTINUE HANDLER FOR NOT FOUND RETURN ret_val;
     LOOP
          FETCH GROUP NEXT ROW; // next row from group
          [ sql expressions ... ]
     END LOOP;
  END
```

- Can implement median, mode, etc.

```
select sum(price * volume), cust_median(price * volume)
from sales group by product;
```

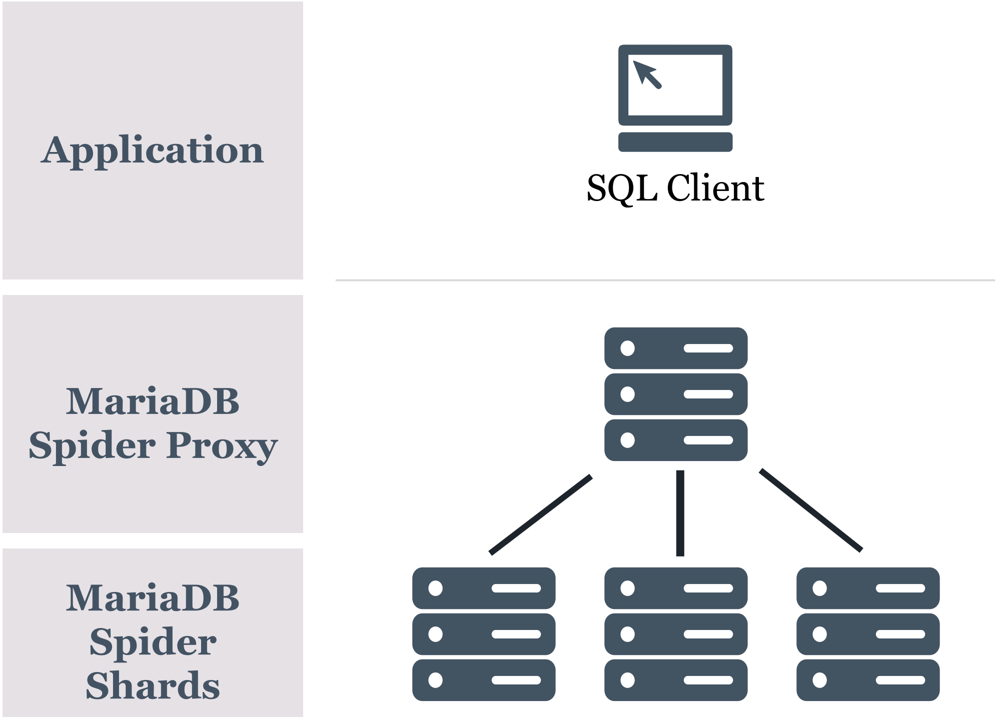


# Storage Engine Enhancements

## Sharding by integration of the Spider Storage Engine

# Sharding by Spider Storage Engine

- The Spider storage engine allows to shard data over multiple MariaDB Server nodes
- A MariaDB Server acting as the Spider Proxy Node
  - Shards are defined by creating partitions
- Partitioned data is stored on the sharding nodes



Application

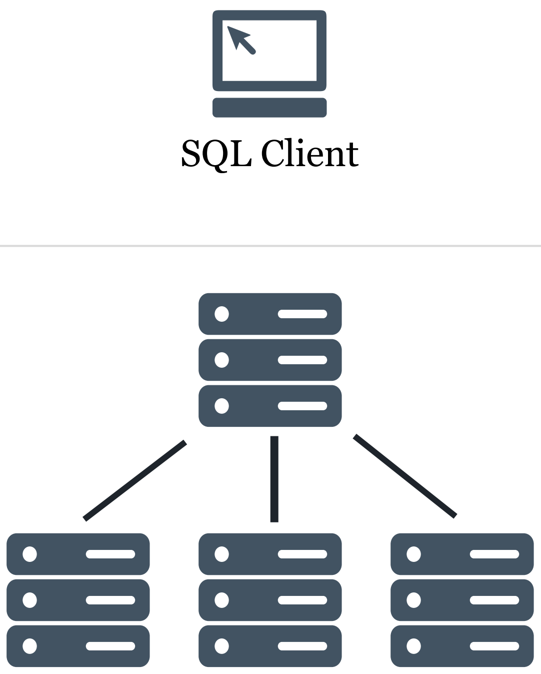
SQL Client

MariaDB Spider Proxy

MariaDB Spider Shards

# Spider Architecture

Application



MariaDB Spider Proxy

Partitions as database links without storing data

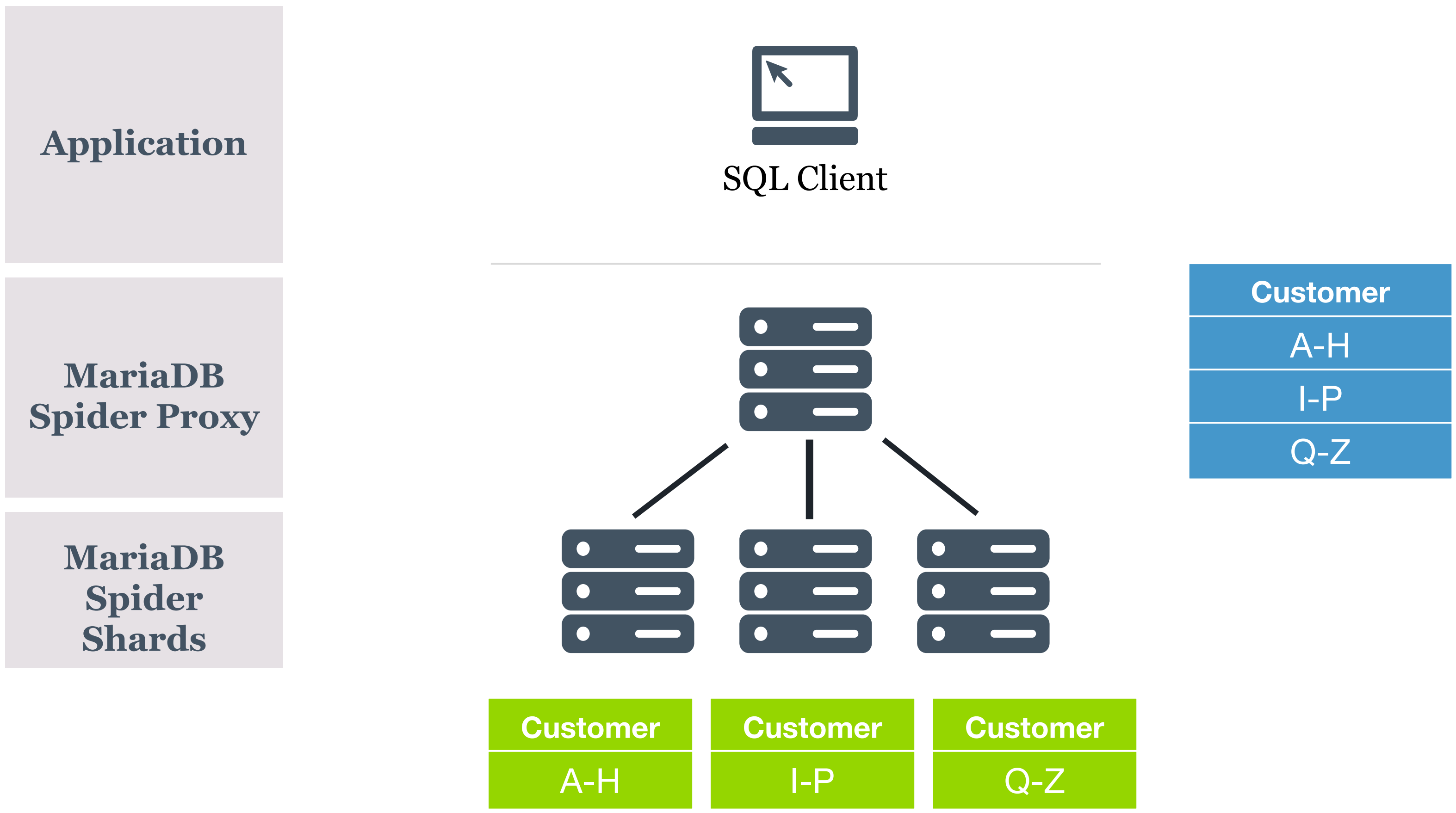
Engine Spider

MariaDB Spider Shards

Partitions sharded as database nodes

Engine InnoDB

# Spider Architecture



Application

MariaDB Spider Proxy

MariaDB Spider Shards

SQL Client

| Customer |
|---|
| A-H |
| I-P |
| Q-Z |

| Customer |
|---|
| A-H |

| Customer |
|---|
| I-P |

| Customer |
|---|
| Q-Z |

# Sharding by Spider Storage Engine

Enhancements in MariaDB Server 10.3

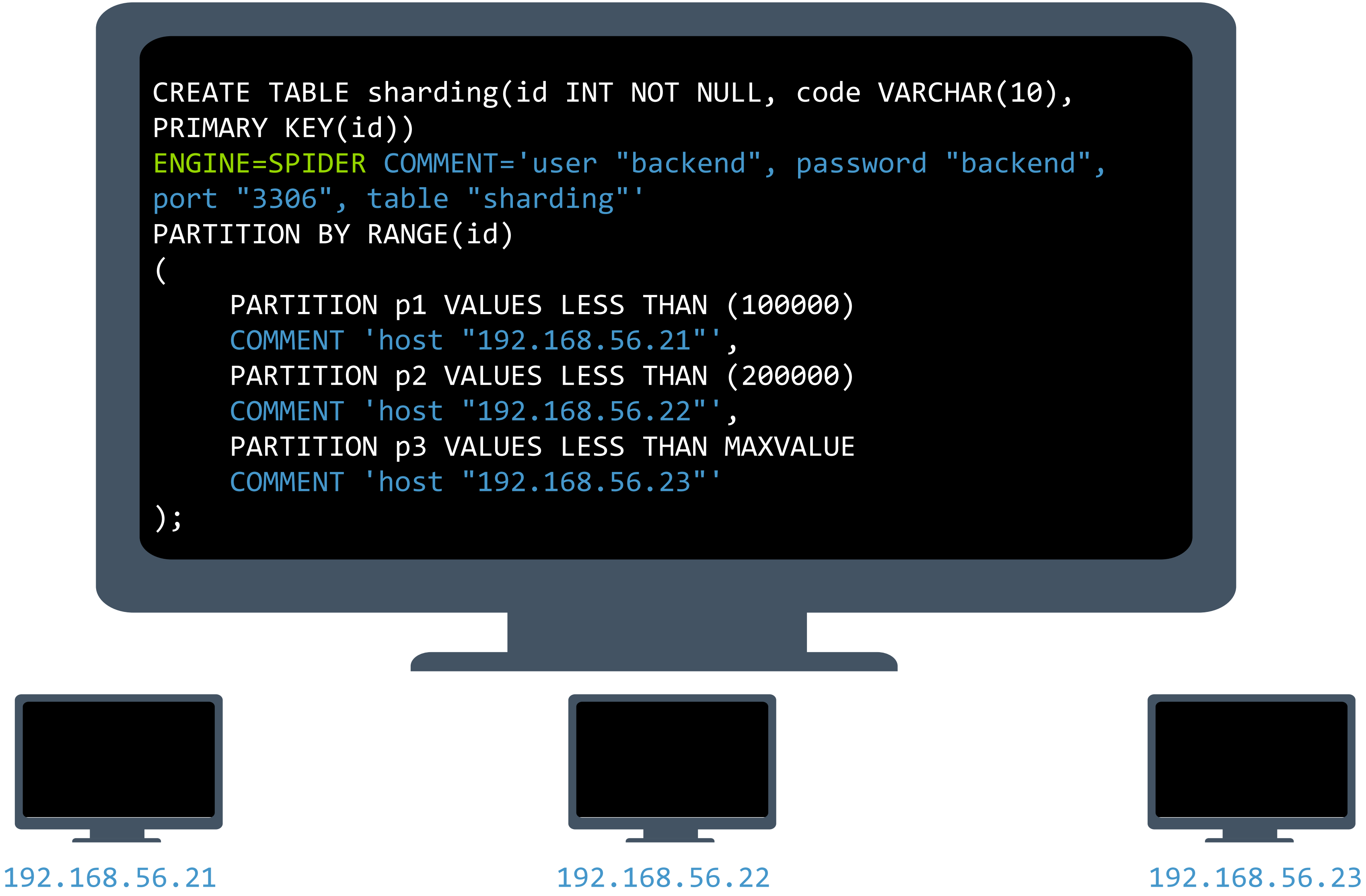
MariaDB Server 10.3. includes the enhancements:

- Integration of the Vertical Partition Engine
  - This allows partitioning by columns
- Engine condition pushdown support in the partition engine to push down the engine condition to the data nodes
- Multi range read support in the partition engine.
- Direct update/delete.
  - This involves pushdown of updates and deletes to the data nodes.
- Full Text Search support in the partition engine.
- Bulk access support in the partition engine.

# Sharding by integration of the Spider Storage Engine

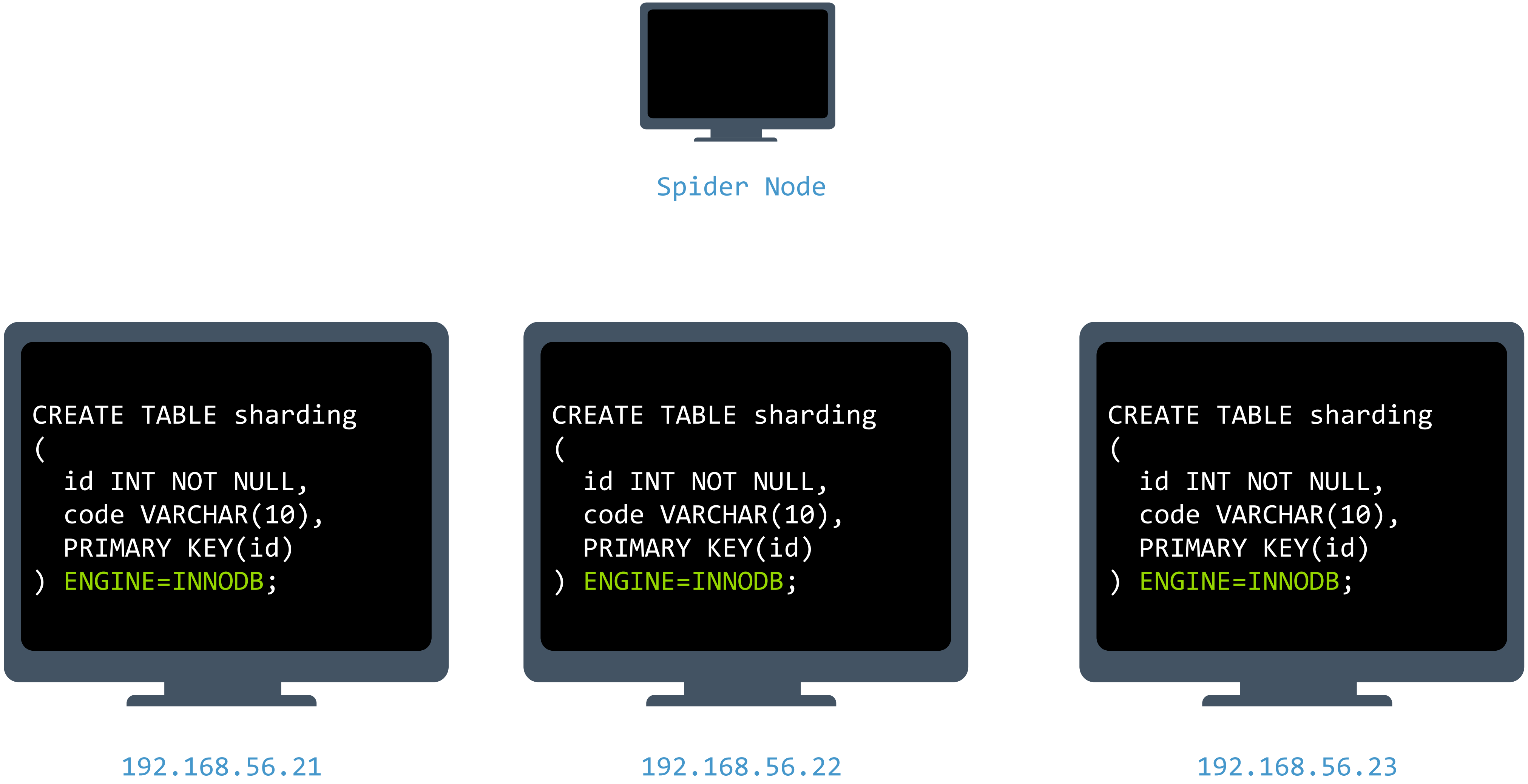
- Auto-increment data type support in the partition engine
- Support for direct aggregation sums, min, max, avg through the partition engine
- Support for child partition pruning in MyISAM Merge tables through the partition engine.
- Option to log result errors
- Options to log stored procedure queries.

# Spider Example



```
CREATE TABLE sharding(id INT NOT NULL, code VARCHAR(10),
PRIMARY KEY(id))
ENGINE=SPIDER COMMENT='user "backend", password "backend",
port "3306", table "sharding"'
PARTITION BY RANGE(id)
(
    PARTITION p1 VALUES LESS THAN (100000)
    COMMENT 'host "192.168.56.21"',
    PARTITION p2 VALUES LESS THAN (200000)
    COMMENT 'host "192.168.56.22"',
    PARTITION p3 VALUES LESS THAN MAXVALUE
    COMMENT 'host "192.168.56.23"'
);
```

# Spider Example



Spider Node

```
CREATE TABLE sharding
(
  id INT NOT NULL,
  code VARCHAR(10),
  PRIMARY KEY(id)
) ENGINE=INNODB;
```

192.168.56.21

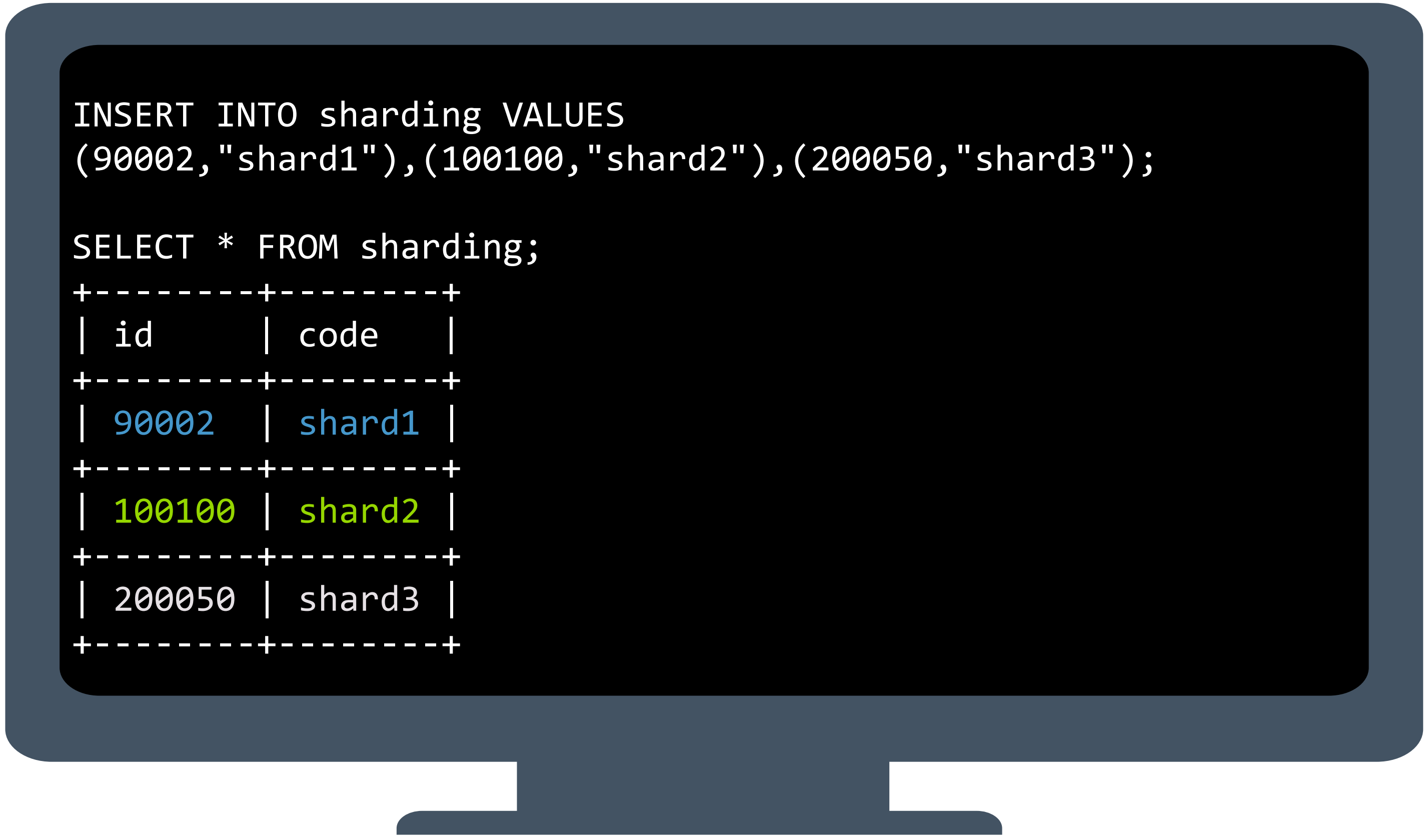
```
CREATE TABLE sharding
(
  id INT NOT NULL,
  code VARCHAR(10),
  PRIMARY KEY(id)
) ENGINE=INNODB;
```

192.168.56.22

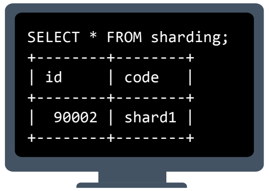
```
CREATE TABLE sharding
(
  id INT NOT NULL,
  code VARCHAR(10),
  PRIMARY KEY(id)
) ENGINE=INNODB;
```

192.168.56.23

# Spider Example

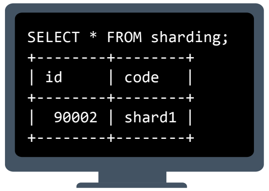


```
INSERT INTO sharding VALUES
(90002,"shard1"),(100100,"shard2"),(200050,"shard3");

SELECT * FROM sharding;
+--------+--------+
| id     | code   |
+--------+--------+
| 90002  | shard1 |
+--------+--------+
| 100100 | shard2 |
+--------+--------+
| 200050 | shard3 |
+--------+--------+
```



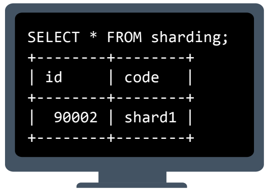
```
SELECT * FROM sharding;
+--------+--------+
| id     | code   |
+--------+--------+
|  90002 | shard1 |
+--------+--------+
```

192.168.56.21



```
SELECT * FROM sharding;
+--------+--------+
| id     | code   |
+--------+--------+
|  90002 | shard1 |
+--------+--------+
```

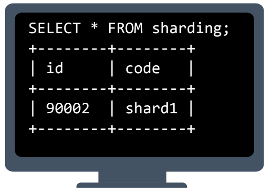
192.168.56.22



```
SELECT * FROM sharding;
+--------+--------+
| id     | code   |
+--------+--------+
|  90002 | shard1 |
+--------+--------+
```

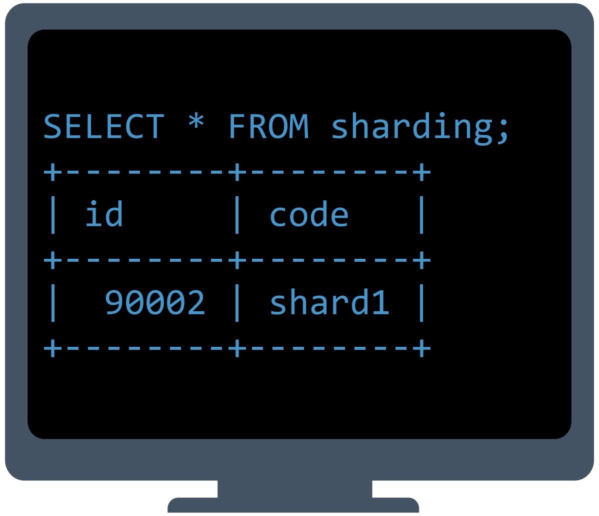
192.168.56.23

# Spider Example



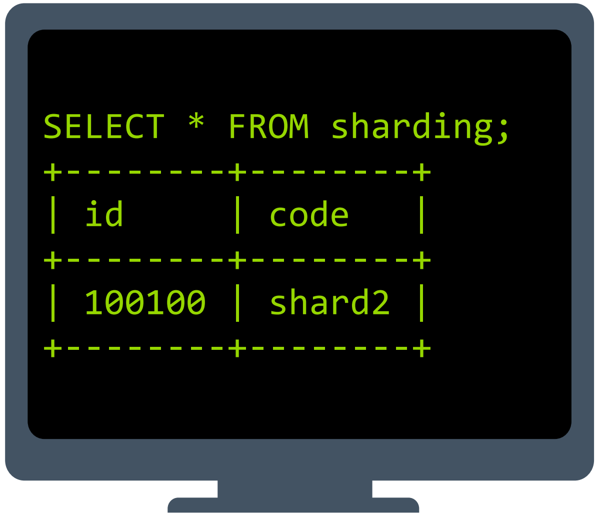
```
SELECT * FROM sharding;
+--------+--------+
| id     | code   |
+--------+--------+
| 90002  | shard1 |
+--------+--------+
```

Spider Node



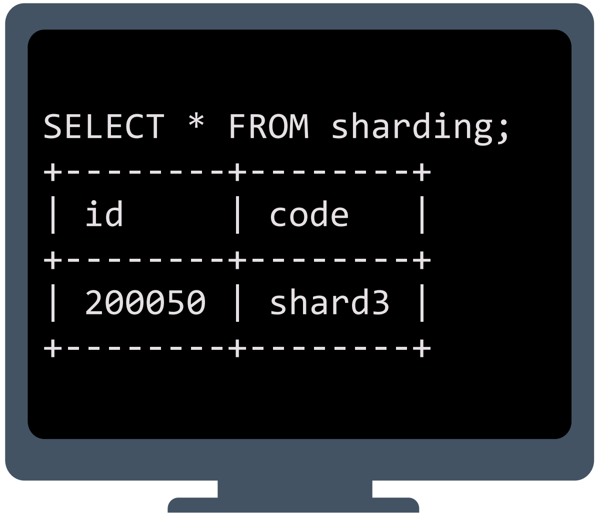
```
SELECT * FROM sharding;
+--------+--------+
| id     | code   |
+--------+--------+
|  90002 | shard1 |
+--------+--------+
```

192.168.56.21



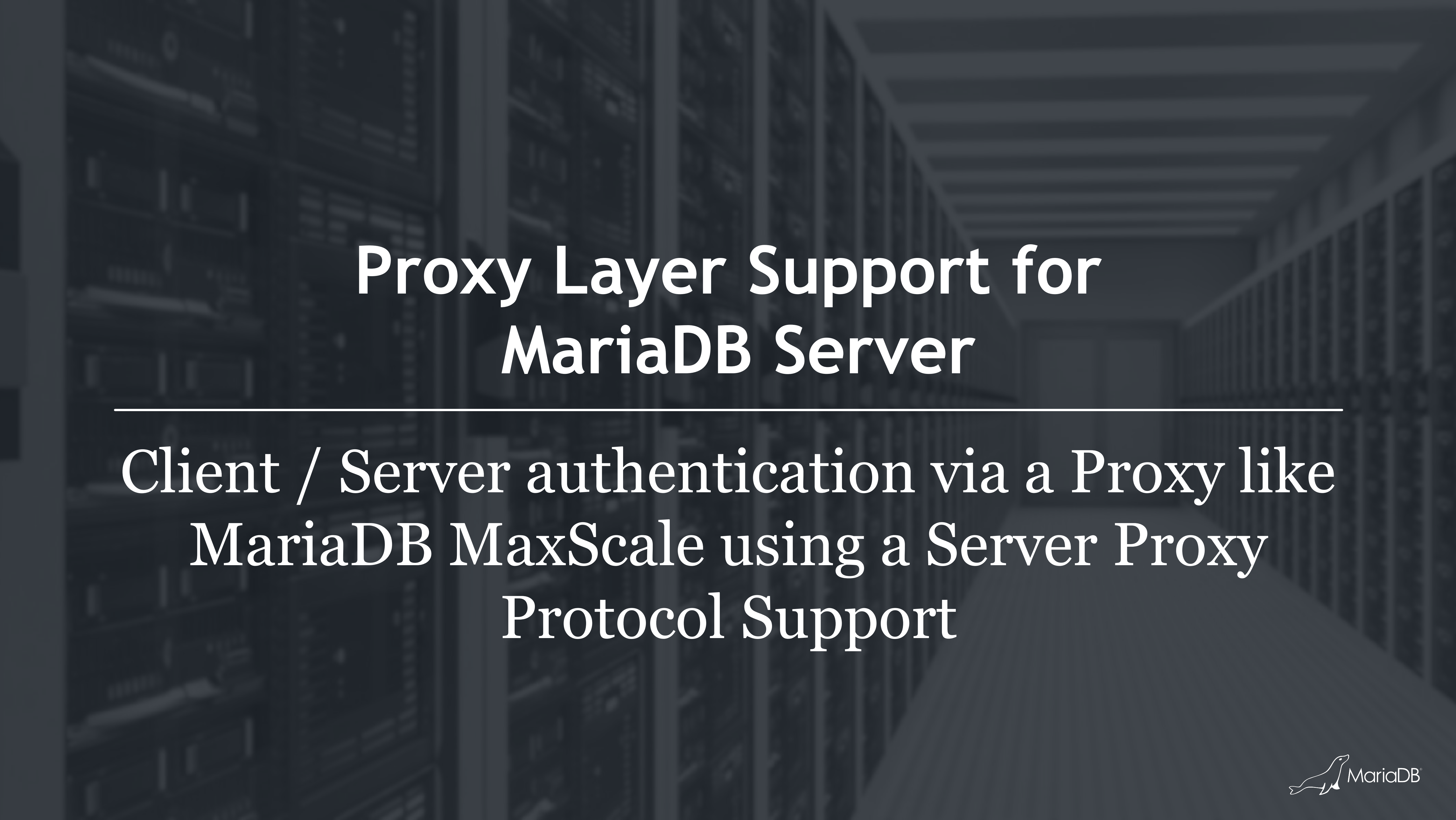
```
SELECT * FROM sharding;
+--------+--------+
| id     | code   |
+--------+--------+
| 100100 | shard2 |
+--------+--------+
```

192.168.56.22



```
SELECT * FROM sharding;
+--------+--------+
| id     | code   |
+--------+--------+
| 200050 | shard3 |
+--------+--------+
```

192.168.56.23



# Proxy Layer Support for MariaDB Server

Client / Server authentication via a Proxy like MariaDB MaxScale using a Server Proxy Protocol Support

# Proxy Layer Support for MariaDB Server

- MariaDB Server 10.3 allows a client to connect to the Server via a proxy without the need to define user privileges based on the host of the proxy
  - The proxy protocol allows a proxy to provide the client IP address to the server
  - This simplifies the user management for clients, which can connect directly to the Server and via Proxy.
  - When introducing a proxy layer, client privileges do not need to be changed
- When using the Audit Plugin and other logs, the logged client IP address is now the real client IP and not the IP from the proxy anymore.
- A new parameter controls the host IP address, which is allowed to use the proxy protocol

# Other Enhancements

Performance, removed limitations and more

- ADD INSTANT COLUMN for InnoDB
  - needs to be the last column
- Set lock wait timeout per statement
  - select ... for update [wait [n] | no_wait]
  - lock table ... [wait [n] | no_wait]
- DELETE statement can delete from the table that is used in a subquery in the WHERE clause
- UPDATE statements can use same source and target
- Compressed Columns

MariaDB®

# Thank you