ORACLE

# MySQL Enterprise Edition JavaScript stored programming

Webinar du 20 Mai 2025

**Emmanuel COLUSSI**

MySQL Solution Architect EMEA

Oracle MySQL GBU

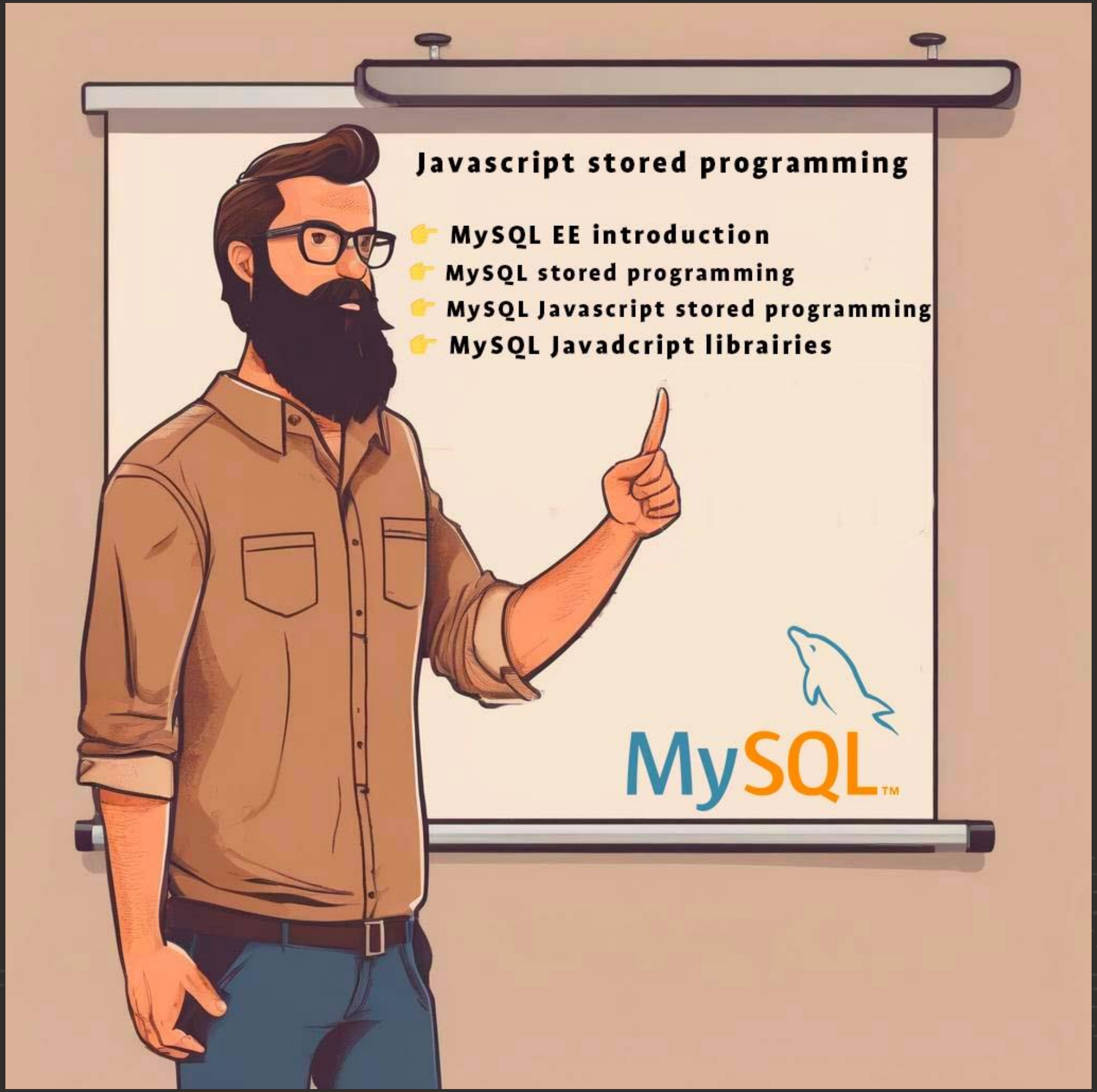
# Speaker

Emmanuel COLUSSI

MySQL Solution Architect EMEA

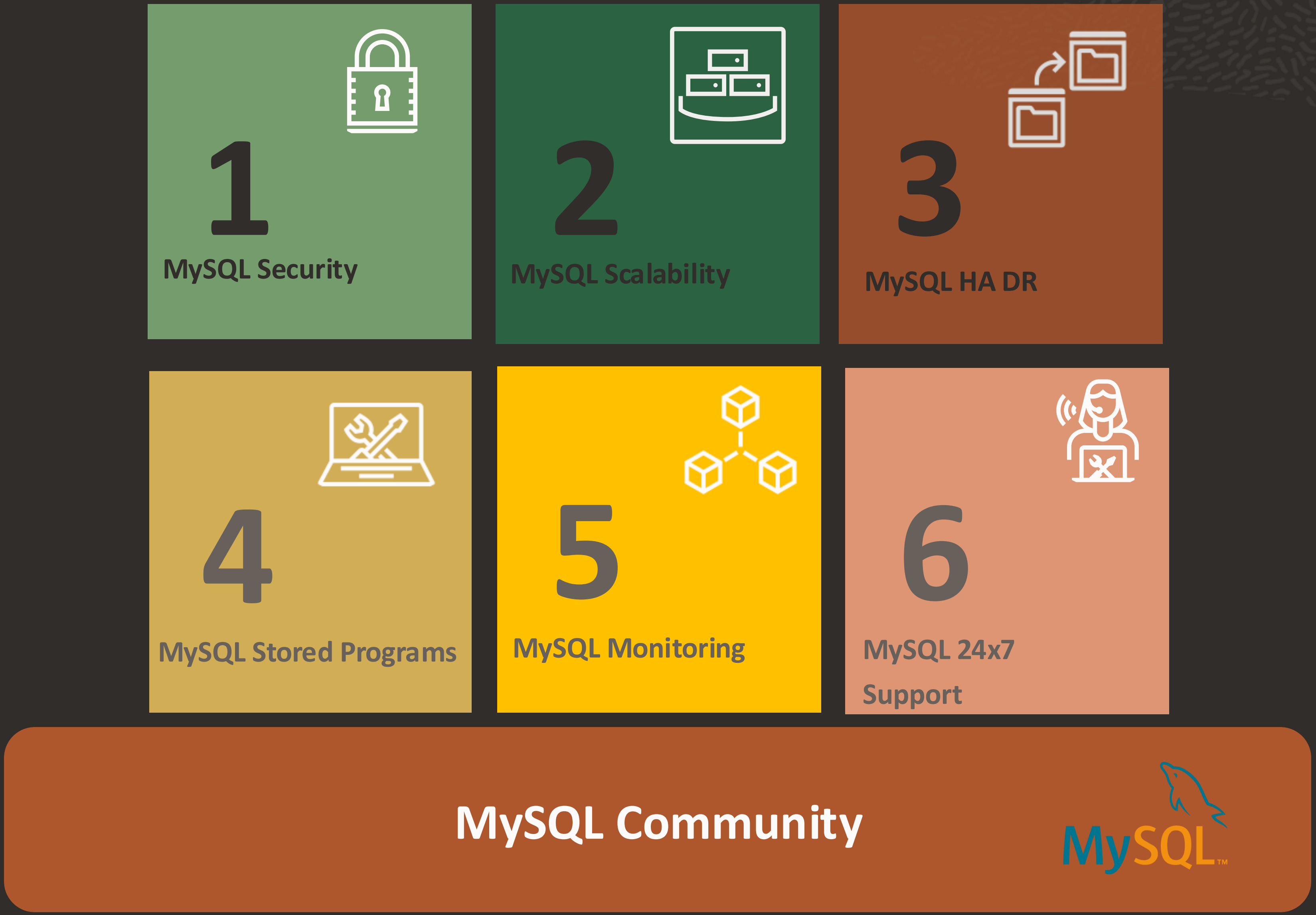
# Agenda

- MySQL  EE introduction
- MySQL SQL stored programming
- MySQL JavaScript stored programming
- MySQL JavaScript libraries


Javascript stored programming
MySQL EE introduction
MySQL stored programming
MySQL Javascript stored programming
MySQL Javadcript librairies
MySQL

# MySQL Enterprise

# MySQL Community VS  MySQL Enterprise



1 MySQL Security

2 MySQL Scalability

3 MySQL HA DR

4 MySQL Stored Programs

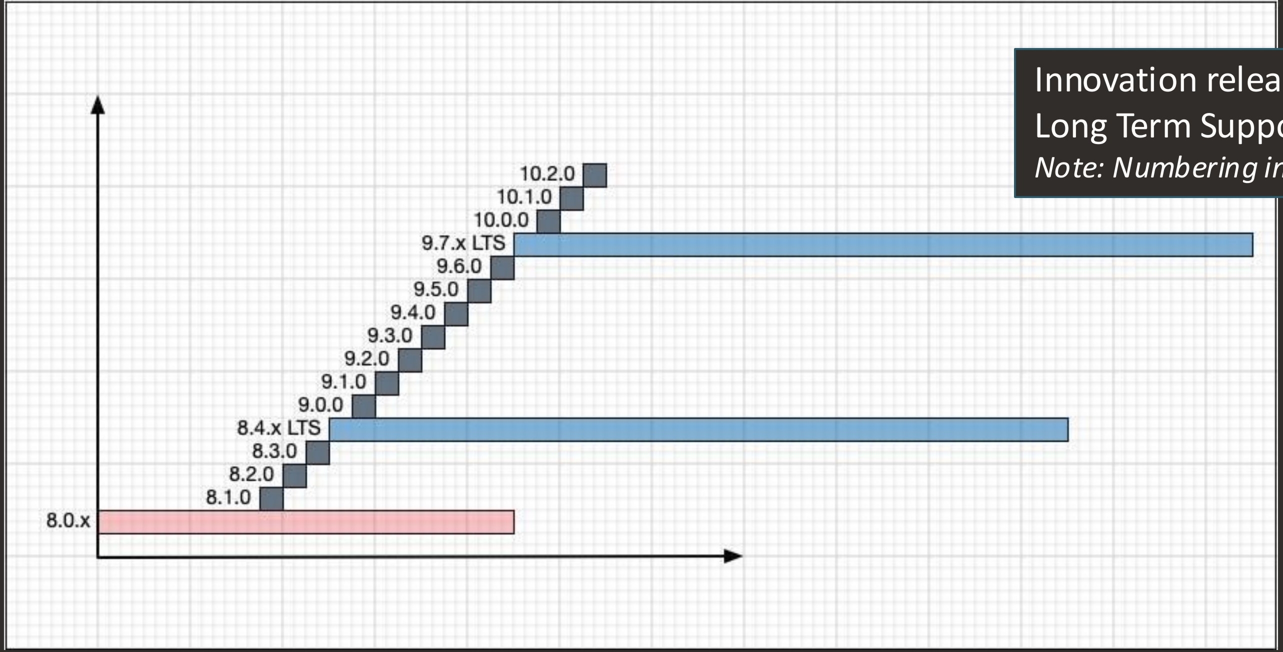
5 MySQL Monitoring

6 MySQL 24x7 Support

MySQL Community

# NEW: MySQL LTS & Innovation Releases

MySQL Community and Enterprise



Innovation releases (grey)
Long Term Support (blue)
*Note: Numbering in the slide is an example and may change*

MySQL Long-Term Support (LTS) Releases

- bugfix & security patches only
- backwards compatibility
- support lifecycle: 5y premier + 3y extended
- Include tools (like MySQL Shell)

MySQL Innovation Releases

- leading-edge innovations
- will likely released every quarter
- Include tools (like MySQL Shell)
- Connectors released as Innovation only

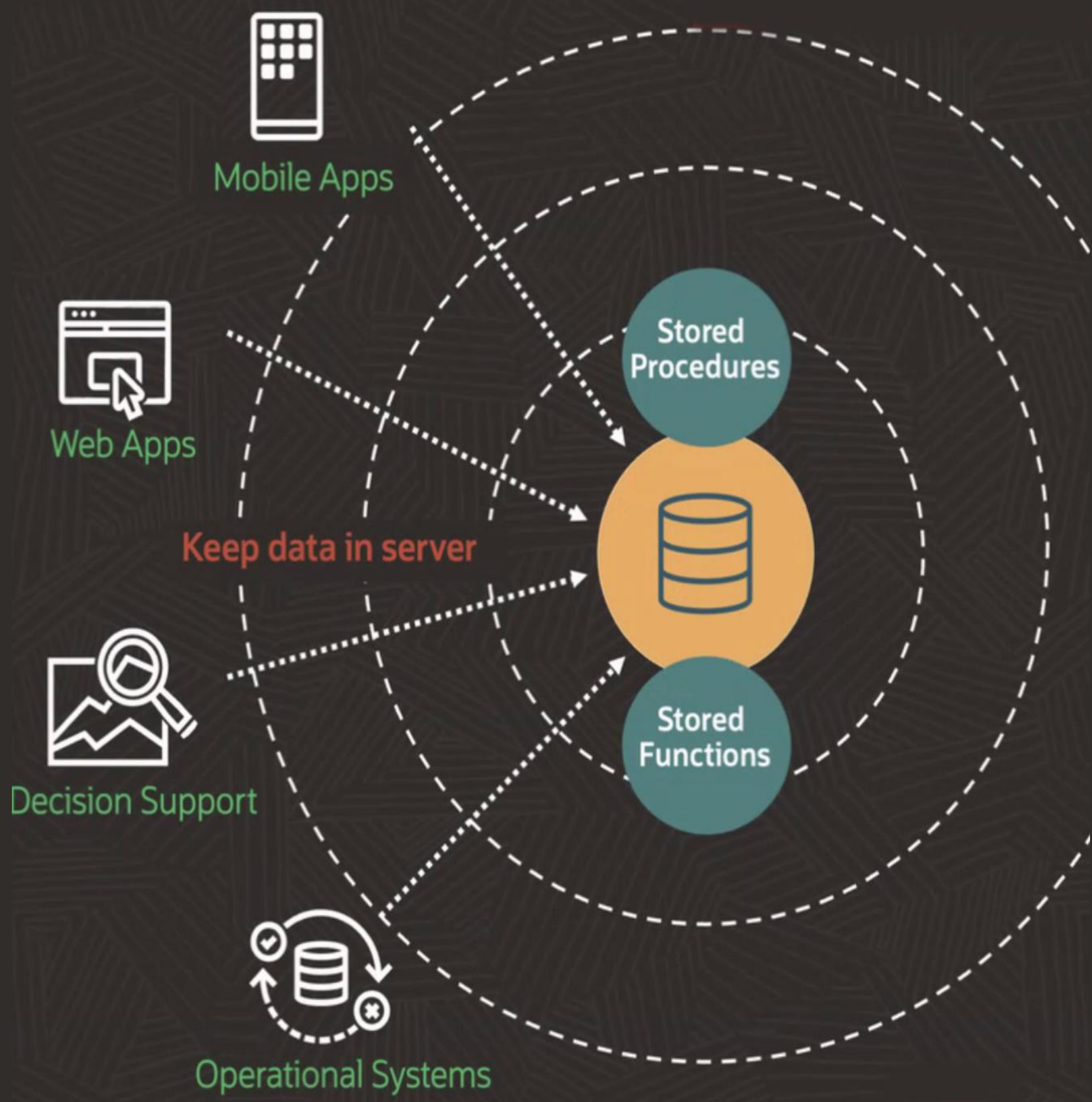
# MySQL Enterprise Stored Programs: Keep Data in MySQL

Stored Programs : Lower costs, reduces complexity, and improves security

Execute JavaScript Stored Procedures and Stored Functions via GraalVM

Handle data-intensive app functionality in stored programs

- Minimize data movement
- Reduce cost
- Improve Security
- Simplify complex ETL → ELT



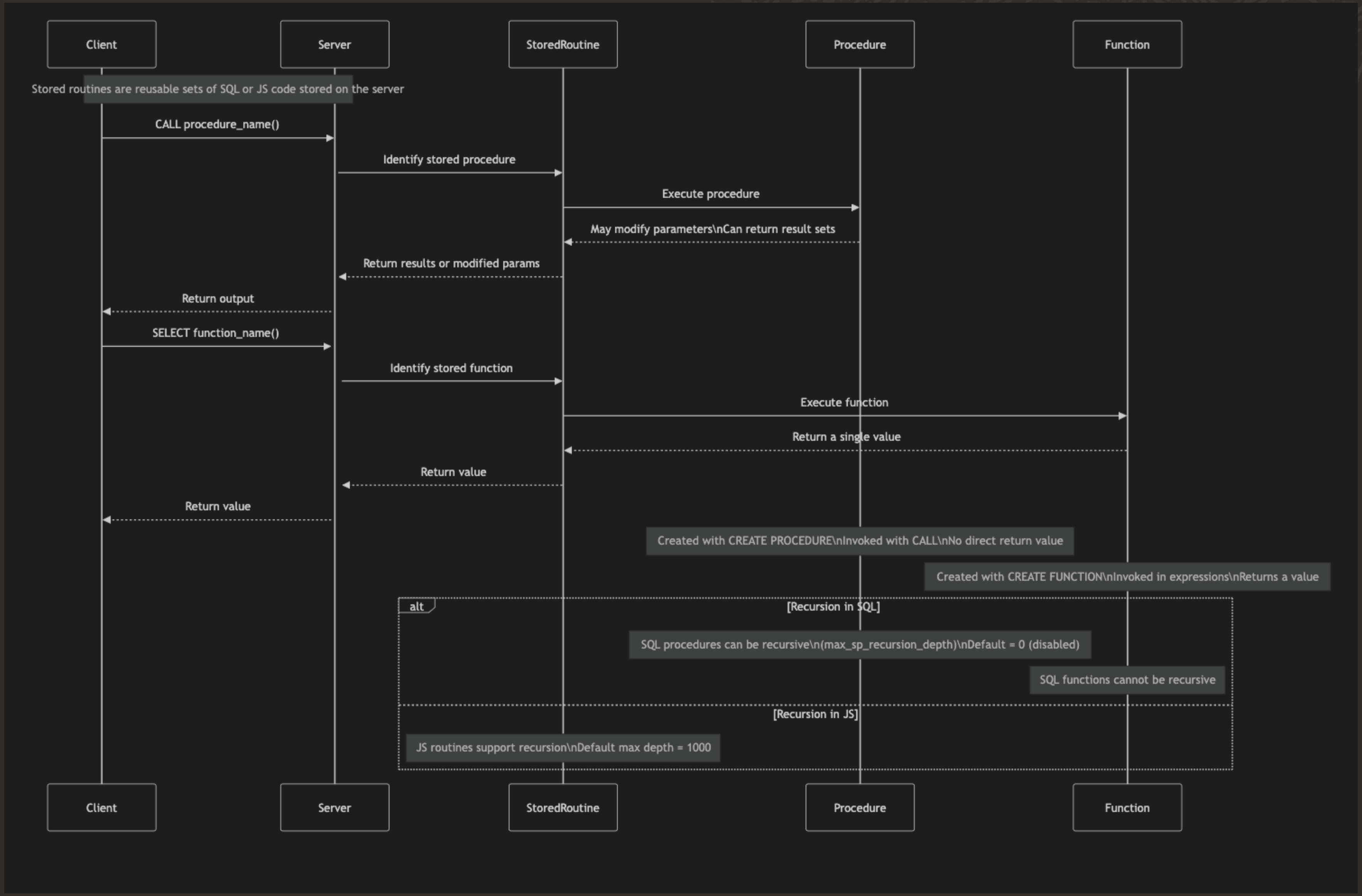
# MySQL SQL stored programming

# MySQL stored routines

- A stored routine is a reusable set of SQL or JavaScript statements stored on the server
    - Clients invoke the routine instead of sending individual statements repeatedly
- Stored Procedures
    - Created using **`CREATE PROCEDURE`**, invoked with **`CALL`**
    - No return value, but can modify parameters for output
    - Can generate result sets for client consumption
- Stored Functions
    - Created using **`CREATE FUNCTION`**, used like built-in functions
    - Invoked in expressions and return a value
    - Associated with a specific database
    - SQL stored functions cannot be recursive
- Recursion Rules
    - SQL stored procedures can be recursive, but recursion is disabled by default (`max_sp_recursion_depth`)
    - JavaScript stored routines support recursion
    - Default maximum depth: 1000

# MySQL stored routines

## Flow



Client | Server | StoredRoutine | Procedure | Function

Stored routines are reusable sets of SQL or JS code stored on the server

CALL procedure_name()

Identify stored procedure

Execute procedure

May modify parameters\nCan return result sets

Return results or modified params

Return output

SELECT function_name()

Identify stored function

Execute function

Return a single value

Return value

Return value

Created with CREATE PROCEDURE\nInvoked with CALL\nNo direct return value

Created with CREATE FUNCTION\nInvoked in expressions\nReturns a value

alt [Recursion in SQL]

SQL procedures can be recursive\n(max_sp_recursion_depth)\nDefault = 0 (disabled)

SQL functions cannot be recursive

[Recursion in JS]

JS routines support recursion\nDefault max depth = 1000

Client | Server | StoredRoutine | Procedure | Function

# Stored procedure security

- There are specific privileges for routines
  - `CREATE ROUTINE, ALTER ROUTINE, EXECUTE`
- SQL SECURITY define the scope of the stored object
  - If a definition omits the DEFINER attribute, the default object definer is the user who creates it
- SQL SECURITY DEFINER to execute in definer security context
  - let other users to execute the command without the privilege on referenced objects
- SQL SECURITY INVOKER to execute in invoker security context
  - let other users to execute the command only on objects where they have the privilege

# Uses of Stored Routines

- Extend functions available to clients
- Centralized client functionality
  - Create a routine in MySQL and make it available to multiple clients
  - Simplified maintenance
  - Improved security
- Create automations
  - Used for triggers
  - Used from event scheduler
- Security
  - Minimize data access, executing with DEFINER privileges
  - Single location processing

# SQL stored procedure example

```
mysql> delimiter //
mysql> CREATE PROCEDURE citycount (IN country CHAR(3), OUT cities INT)
       BEGIN
         SELECT COUNT(*) INTO cities FROM world.city WHERE CountryCode = country;
       END//
Query OK, 0 rows affected (0.01 sec)
mysql> delimiter ;
mysql> CALL citycount('JPN', @cities); -- cities in Japan
Query OK, 1 row affected (0.00 sec)
mysql> SELECT @cities;
+---------+
| @cities |
+---------+
|     248 |
+---------+
1 row in set (0.00 sec)
```

# SQL stored function example

```
mysql> delimiter //
mysql> CREATE FUNCTION hello (s CHAR(20)) RETURNS CHAR(50) DETERMINISTIC
       BEGIN
         RETURN CONCAT('Hello, ',s,'!');
       END//
Query OK, 0 rows affected (0.00 sec)
mysql> delimiter ;
mysql> SELECT hello('world');
+----------------+
| hello('world') |
+----------------+
| Hello, world!  |
+----------------+
1 row in set (0.00 sec)
```

# Stored Routine Metadata

- Query the ROUTINES table in the INFORMATION_SCHEMA database
- Use SHOW CREATE PROCEDURE and SHOW CREATE FUNCTION to view routine definitions
- Use SHOW PROCEDURE STATUS and SHOW FUNCTION STATUS to examine routine characteristics
- **Up to MySQL 8.4**: Stored routines are SQL-only
- Refer to the manual for details on variables and flow control
- **From MySQL 9.0 onwards**: JavaScript support is available
- **Starting with MySQL 9.2**: Create libraries to enhance routine reusability

# Javascript support inside MySQL

## ECMAScript 2023 Compliant

- Runs in **strict mode** by default
- Supports standard built-in objects: Object, Function, Math, Date, String, etc.

## Scoped Execution Environment

- “global” and “globalThis” are supported but scoped to the current routine
- **No access** to filesystem or network from JavaScript code
- **Node.js is not supported**

## Execution Model

- Single-threaded per query
- Async features (Promise, async/await) are **simulated** – may show **non-deterministic behavior**

## Interop with SQL

- Call SQL stored programs from JavaScript
- JavaScript routines can be invoked from SQL procedures, functions, events, and triggers
- Supports recursion up to **1000 levels**

## Developer Convenience

- No need to change DELIMITER – semicolon (;) is handled transparently

# MySQL to JavaScript data types conversion

- Most MySQL data types are supported for MLE stored program input and output arguments, as well as for return data types
  - Automatic conversion between MySQL and JavaScript data types

| MySQL Type | JavaScript Type |
|---|---|
| TINYINT, SMALLINT, MEDIUMINT, INT, BOOL, BIGINT, or SERIAL | If safe: Number; otherwise: String |
| FLOAT or DOUBLE | Number |
| CHAR, VARCHAR, TINYTEXT, TEXT, MEDIUMTEXT, or LONGTEXT | String |
| TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB, BINARY, or VARBINARY | Uint8Array |
| DATE, DATETIME, or TIMESTAMP | Date |
| TIME | String |
| YEAR | Number |
| VECTOR | Float32Array |

- Details here: https://dev.mysql.com/doc/refman/9.3/en/srjs-data-arguments.html

# JavaScript to MySQL Data Type Mapping – Key Considerations

- Dynamic Typing in JavaScript
  - JavaScript is dynamically typed; return types are determined **only at runtime**
  - This differs from SQL's static typing model
- Automatic Type Conversion
  - MySQL **automatically converts** JavaScript return types to compatible MySQL types
  - Refer to the MySQL 9.3 Manual – Data Argument Conversions for full details
  - Examples:
    - **JavaScript Boolean** → MySQL TINYINT, SMALLINT, CHAR, VARCHAR, FLOAT, etc.
    - **JavaScript String** → MySQL TINYINT, SMALLINT, CHAR, VARCHAR, FLOAT, etc.
- Special Notes
  - Infinity and -Infinity are treated as **out-of-range values**
  - NaN results in an **invalid type conversion error**
  - **All rounding** is done using Math.round()
  - **BigInt or non-numeric Strings** cast to FLOAT cause **conversion errors**

# JavaScript time zone support

- JavaScript stored programs use the MySQL session time zone active at their first invocation
  - This time zone remains fixed for the duration of the session
- Changing the session time zone later does not affect already-used (cached) stored programs
- To apply the new time zone:
  - Call **`mle_session_reset()`** to clear the cache
  - Subsequent invocations will use the updated session time zone

# JavaScript override of SQL arguments

- Do not redeclare program arguments using let, var, or const inside JavaScript stored programs
- Redeclaring parameters creates local variables that shadow the original arguments
- This makes the passed-in values inaccessible
- Always use the argument as-is to preserve its original value and behavior

- Example:

```
mysql> CREATE FUNCTION myfunc(x INT)RETURNS INT LANGUAGE JAVASCRIPT AS
$mle$
  var x
  return 2*x
$mle$;
Query OK, 0 rows affected (0.03 sec)

mysql> SELECT myfunc(10);
ERROR 6000 (HY000): MLE-Type> Cannot convert value 'NaN' to INT from MLE in 'myfunc(10)'
```

# SQL/Javascript example: Greatest Common Divisor

## SQL

```
DELIMITER //
CREATE FUNCTION gcd_sql(x int, y int)
RETURNS int DETERMINISTIC
BEGIN
  DECLARE dividend int;
  DECLARE divisor int;
  DECLARE remainder int;

  -- Order arguments for MOD function
  SET dividend := GREATEST(x, y);
  SET remainder := LEAST(x, y);

  WHILE remainder != 0 DO
    SET divisor = remainder;
    SET remainder = MOD(dividend,
divisor);
    SET dividend = divisor;
  END WHILE;

  RETURN divisor;
END
//
DELIMITER ;
```

## JavaScript (like SQL)

```
CREATE FUNCTION gcd_js(x int, y int)
RETURNS int DETERMINISTIC
LANGUAGE JAVASCRIPT
AS $mle$
  let dividend=1;
  let divisor=1;
  let remainder=1;

  // Replicate SQL, but % works fine with
  // first argument > second argument
  dividend = (x > y) ? x : y ;
  remainder = (x < y) ? x : y ;

  while (remainder !== 0) {
    divisor = remainder;
    remainder = dividend % divisor;
    dividend = divisor;
  }

  return divisor;
$mle$
;
```

## JavaScript (optimized)

```
CREATE FUNCTION gcd_js2(arg1 INT, arg2 INT)
RETURNS INT DETERMINISTIC
LANGUAGE JAVASCRIPT
AS $mle$
  let gcd_js_rec = (a, b) => b ?
     gcd_js_rec(b, a % b) : Math.abs(a)

  return gcd_js_rec(arg1, arg2)
$mle$;



/*
NOTE: no DELIMITER manual
handling:
  ';' DELIMITER is handled
transparently
*/
```

# Javascript SQL API

- Supported Top-Level Objects:
  - **`Column`** – Metadata for result set columns
  - **`Row`** – Represents a single row in a result set
  - **`PreparedStatement`** – Handles execution of prepared SQL statements
  - **`SqlExecute`** – Executes simple SQL via execute()
  - **`SqlResult`** – Result set returned by an SQL statement
  - **`Warning`** – Captures warnings raised during statement execution
  - **`Session`** – Represents the MySQL user session
    - Supports startTransaction(), commit(), rollback()
    - Provides access to MySQL session variables
- Additional Features:
  - getFunction() / getProcedure() return JavaScript Function objects for MySQL stored routines
  - Access to schema, table, row, and column objects
  - MySQL namespace includes:
    - SQL built-in functions
    - `SqlError` (equivalent to MySQL SIGNAL)
- Note: SQL API is available only in JavaScript stored procedures (not supported in JavaScript stored functions)

# Example of SQL inside JavaScript

```
mysql> CREATE PROCEDURE cities_1million(arg1 CHAR(3)) LANGUAGE JAVASCRIPT
AS $mle$
  console.clear()
  let s = session.sql('SELECT Name FROM world.city WHERE Population > 1000000 AND CountryCode = "' + arg1 + '"')
  let res = s.execute()
  let row = res.fetchOne()

  while(row) {
    console.log(row.toArray())
    row = res.fetchOne()
  }
$mle$;

mysql> CALL cities_1million('ITA');
Query OK, 0 rows affected (0.0030 sec)

mysql> SELECT mle_session_state("stdout") AS 'STDOUT';
+---------------------+
| STDOUT              |
+---------------------+
| Roma
Milano
Napoli
…
```

# JavaScript session information

- If not cleared, console output is appended to existing session state on each run
- The MLE component provides a number of loadable functions for working with MLE user sessions
- mle_session_reset()
  - Clears current MLE session state
  - Can reset all or specific parts: "stdout", "stderr", "output"
- mle_session_state()
  - Returns session info about the most recently executed MLE stored program
  - Includes accumulated output from console.log() / console.error()
- mle_set_session_state()
  - Defines conversion rules for mapping MySQL INTEGER and DECIMAL types to JavaScript values

Example

```
mysql> CREATE PROCEDURE helloword_js()
LANGUAGE JAVASCRIPT
AS $mle$
  console.log("Hello World")
$mle$;

mysql> CALL helloword_js;
Query OK, 0 rows affected (0.0072 sec)

mysql> SELECT mle_session_state("stdout")
AS 'STDOUT’;
+---------------+
| STDOUT        |
+---------------+
| Hello World
|
+---------------+
```

# SqlError Object

- You can create an SqlError using the constructor shown here:

```
new SqlError(
sql_state: Number,
message: String,
error_number: Number
)
```

- When an SqlError is **thrown**, an error is raised in MySQL similar to how one is raised by a SIGNAL statement, like

```
mysql> CREATE PROCEDURE test_catch_throw_signal() LANGUAGE JAVASCRIPT
AS $mle$
try { throw new mysql.SQLError(45000, 'Some error', 1001) }
catch (e) { console.log(e) }
$> $$;
Query OK, 0 rows affected (0.02 sec)

mysql> CALL test_catch_throw_signal();
Query OK, 0 rows affected (0.04 sec)

mysql> SELECT mle_session_state("stdout")\G
*************************** 1. row ***************************
org.graalvm.polyglot.nativeapi.PolyglotNativeAPI$CallbackException: SQL-CALLOUT:
Error code: 1001 Error state: 45000 Error message: `Some error`
```

# Javascript Error Handling

- An SQL statement that causes errors that are not handled within the stored program passes them back to the client
- Error can be handled using one or more “**try ... catch**" blocks

```
mysql> CREATE PROCEDURE jssp_catch_errors(IN query VARCHAR(200)) LANGUAGE JAVASCRIPT
AS $mle$
try { var result = session.sql(query).execute()
} catch (e) { console.error("\nJS Error:\n" + e.name + ":\n" + e.message)
}
$mle$;

mysql> CALL jssp_catch_errors("SELECT * FROM bogus");
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT mle_session_state('stderr')\G
*************************** 1. row ***************************
mle_session_state('stderr'):
JS Error:
org.graalvm.polyglot.nativeapi.PolyglotNativeAPI$CallbackException:
SQL-CALLOUT: Error code: 1146 Error state: 42S02 Error message: Table 'test.bogus' doesn't
exist
```

# MySQL JavaScript libraries

# JavaScript libraries

- One of the benefits of JavaScript is the option to create and reuse libraries
  - Remove code duplication
  - Better code maintainability
  - Better code encapsulation
  - Performance improvement
- JavaScript language support in MySQL conforms to the ECMAscript 2023 standard
- Libraries can be
  - Created inside MySQL, with specific syntax
  - Include other libraries available in the same MySQL instance
  - Loaded from a file, adding an SQL specific header and footer
- To create a library use CREATE LIBRARY Statement, like

```
mysql> CREATE LIBRARY IF NOT EXISTS jslib.lib1 LANGUAGE JAVASCRIPT
AS $mle$
  export function f(n) {
    return n * 2
  }
$mle$;
```

# Import libraries inside a store program

- A new USING clause was introduced for stored functions & procedures
  - Example

```
mysql> CREATE FUNCTION foo(n INTEGER) RETURNS INTEGER LANGUAGE JAVASCRIPT
USING (jslib.lib1 AS mylib, jslib.lib2 AS yourlib)
AS $mle$
  return mylib.f(n) + yourlib.g(n)
$mle$;
```

- JavaScript syntax is checked at library creation time, as shown here:

```
mysql> CREATE LIBRARY IF NOT EXISTS jslib.lib3 LANGUAGE JAVASCRIPT
AS $mle$
  export function f(n) {
    return n $ 2
  }
$mle$;
ERROR 6113 (HY000): JavaScript> SyntaxError: lib3:3:17 Expected ; but found $
        return n $ 2
                 ^
```

# How to import external libraries

- External libraries, written in ECMAscript 2023 standard can be imported, enclosing the library content between library creation statements, for example
  1. Rename the **`my_library.js`** file to **`my_library.sql`**
  2. Edit the `mysqlibray.sql` file enclosing between create library statements

     ```
     CREATE LIBRARY mydb.my_library LANGUAGE JAVASCRIPT
     AS $mle$
     // ... content of the .js or .mjs file
     $mle$;
     ```

  3. Load the `mysqlibray.sql` file like usual

     ```
     mysql -uroot -p < my_library.sql
     ```

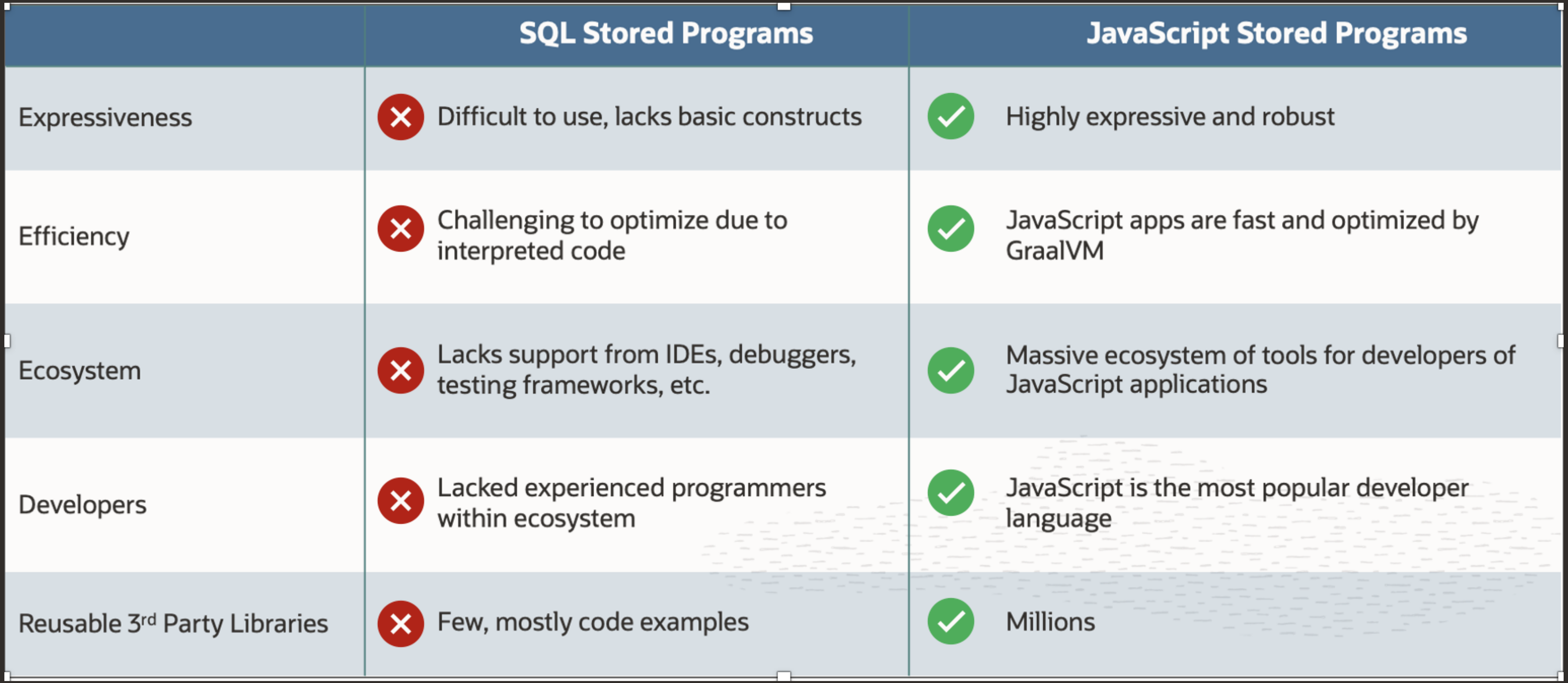
- Currently there is no automatic resolution of dependencies, so you need to encode all the dependencies in the same library
  - use webpack, rollup or a similar tool to create a bundle that include all

# The power of libraries: examples

- Expand math calculation for additional functions or math data types, e.g.
  - Advanced Math for statistics and complex calculations
  - Vectors for genAI
- Add Machine Learning algorithms
  - To extract new value form existing data inside MySQL, without export/import
- Use a validator plugin to prevent invalid values from ever being stored in the DB, for more secure applications
- Use a date/time library to handle expressions like "tomorrow" or "last Friday", or calculate the CW of a date and make applications more user/developer friendly
- Manipulate URIs to remove the passwords or split URIs into user, host, parameters and store separately in columns and simplify developers
  - And generated columns may expose these directly in a table
- And much more

# Key takeaways

# Key takeaways



| | SQL Stored Programs | JavaScript Stored Programs |
|---|---|---|
| Expressiveness | ✗ Difficult to use, lacks basic constructs | ✓ Highly expressive and robust |
| Efficiency | ✗ Challenging to optimize due to interpreted code | ✓ JavaScript apps are fast and optimized by GraalVM |
| Ecosystem | ✗ Lacks support from IDEs, debuggers, testing frameworks, etc. | ✓ Massive ecosystem of tools for developers of JavaScript applications |
| Developers | ✗ Lacked experienced programmers within ecosystem | ✓ JavaScript is the most popular developer language |
| Reusable 3rd Party Libraries | ✗ Few, mostly code examples | ✓ Millions |

# Resources

- MySQL Stored programming
  - https://dev.mysql.com/doc/refman/9.3/en/stored-objects.html
- MySQL Javascript store programming
  - https://dev.mysql.com/doc/refman/9.3/en/stored-routines-js.html
- MySQL JavaScript examples from manual
  - https://dev.mysql.com/doc/refman/9.3/en/srjs-examples.html
- MySQL JavaScript example: MySQL vector operations
  - https://lefred.be/content/mysql-vector-datatype-create-your-operations-part-1/
- MySQL JavaScript example: UUID
  - https://blogs.oracle.com/mysql/post/javascript-support-in-mysql-the-uuid-example

# Thank you

**Emmanuel COLUSSI**

emmanuel.colussi@oracle.com

ORACLE