ORACLE

JavaScript for MySQL HeatWave (Limited Availability)

Use JavaScript stored programs in MySQL HeatWave

September 2023

Purpose statement

This document provides an overview of the JavaScript for MySQL HeatWave feature. It is intended solely to help you assess the benefits of MySQL HeatWave JavaScript support and to plan your I.T. projects.

Disclaimer

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle software license and service agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced or distributed to anyone outside Oracle without prior written consent of Oracle. This document is not part of your license agreement, nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

This document is for informational purposes only and is intended solely to assist you in planning for the implementation and upgrade of the product features described. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in this document remains at the sole discretion of Oracle. Due to the nature of the product architecture, it may not be possible to safely include all features described in this document without risking significant destabilization of the code.

Benchmark queries are derived from the TPC-H benchmark, but results are not comparable to published TPC-H benchmark results since they do not comply with the TPC-H specification.

Table of contents

Executive Summary

MySQL HeatWave is a fully managed cloud database service, powered by the built-in HeatWave in-memory query accelerator. It's the only cloud database service that combines transactions, real-time analytics across data warehouses and data lakes, and machine learning in one MySQL Database—without the complexity, latency, risks, and cost of ETL duplication. It's available on OCI, AWS, Azure, and in customers' data centers with OCI Dedicated Region. MySQL HeatWave delivers the best performance and price-performance in the industry for data warehousing.

MySQL HeatWave now includes rich procedural programming capability directly inside the database, further enabling the user to cut down on data movement cost in favor of server-side solutions. "**JavaScript for MySQL HeatWave**" is a new feature available exclusively in MySQL HeatWave. It allows users to write JavaScript stored functions and procedures in the server that are executed via GraalVM. The JavaScript functions and procedures can manipulate existing MySQL data irrespective of the underlying storage engine, i.e., InnoDB, MyISAM, or HeatWave, all work transparently.

Users can now re-organize applications and move the data-intensive complex operations closer to their data, this reduces the cloud egress cost and the effort required to maintain data pipelines. In addition, it improves end-to-end application performance and security by eliminating the need for client-server data movement.

Challenges

Even with rich transaction processing, analytics, and machine learning inside the database, complex and rapidly evolving data-intensive applications still force the user to move large amount of data into the client side of applications. This is done to access the rich procedural programming language eco-system not available inside the database. Enabling the same capability in the database has some practical challenges:

Procedural SQL Limitations

MySQL allows stored programs in SQL procedural-dialect "Compound Statements" [4]. This allows users to deploy server-side programs but comes with limitations. SQL stored programs are interpreted and do not take advantage of compiler optimizations. Furthermore, the SQL dialects lack basic features compared to modern language runtime, such as user-defined types, containers (arrays, maps), and functional programming constructs. This makes it hard for the user to implement complex logic.

Development Ecosystem

Only allowing procedural language runtime inside the database is not sufficient, the challenge is to give developers the freedom to use the development ecosystem that comes with the language runtime. The development ecosystem may include tools such as 3rd party package managers, debuggers, editors, testing frameworks, etc…

Data Access API

Accessing and manipulating database data is central to procedural language integration. Introducing a new data access API to the developer community introduces a steep learning curve and hinders adoption. Reusing interfaces such as existing database client-server connectors and ORMs does not directly map to the needs of server-centric data processing.

Security

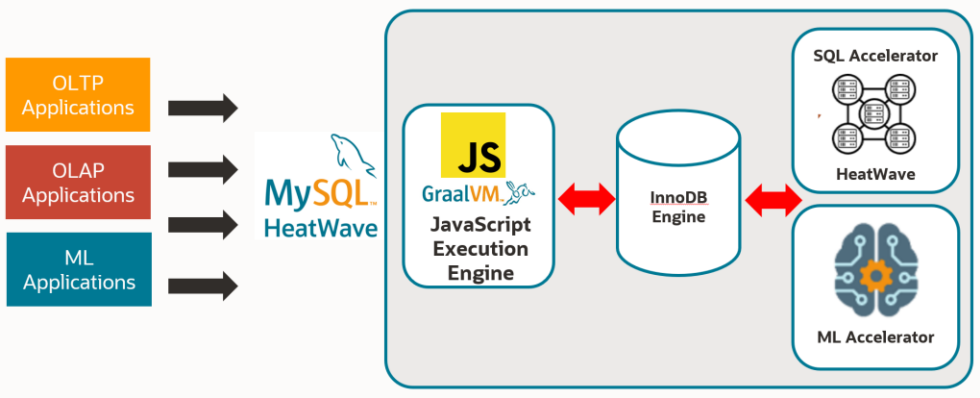
Adding a new execution engine in the database server means expanding the attack surface for the database against new vulnerabilities. It also means preventing unnecessary access to valuable compute and memory resources by the procedural code that would have been otherwise used by the database.

Introducing "JavaScript for MySQL HeatWave"

To address the challenges discussed above, we are introducing support for JavaScript stored programs in MySQL Heatwave (currently in limited availability). Users can now express rich procedural logic inside the database and access their MySQL datasets seamlessly without incurring ETL costs. The JavaScript runtime is integrated via GraalVM, where the user can use all GraalVM's Enterprise Edition (EE) features such as compiler optimizations, performance, and security features at no additional cost.

In the initial release the feature supports

- JavaScript language based on ECMAScript 2023
- MySQL data types such as all variations of integers, floating point, temporal and VARCHAR, CHAR types with full utf8mb4 support
- Data access API based on MySQL Shell JavaScript XDevAPI



Why JavaScript

JavaScript is one of the most popular programming languages [1]. As of 2023, more than 98% of all the websites use JavaScript [5]. Apart from support in all major web browsers, JavaScript is widely used in server-side runtime such as Node.js[7] and Deno[8]. The language has a collection of over a million reusable 3rd party packages in "npm" alone, the package manager is used by tens of millions of developers [6].

With JavaScript, developers can take advantage of the weakly-typed procedural language inside the server. The JavaScript built-in library includes many commonly used operations and data structures that make implementation easy and expressive. The developer can also reuse available 3rd party libraries without reimplementing everything from scratch.

GraalVM

The feature uses the Enterprise edition of GraalVM [2]. GraalVM is an Oracle compiler ecosystem that includes JDK, language implementation such as JavaScript, R, Python, Ruby and Java. It includes just-in-time (JIT) and ahead-of-time (AOT) compilation technology. It also provides a fully managed virtual machine with sandboxing capability and tooling support. More details below:

Graal.JS

GraalVM has its own JavaScript implementation based on the ECMAScript 2023 standard. The language implementation is competitive in terms of performance [9] even though it is implemented using GraalVM's Polyglot framework which focuses on extending language support in the same VM.

Optimizations

GraalVM comes with its own state-of-the-art compiler optimizations including aggressive inlining and partial escape analysis. It also provides a profile guided just-in-time (JIT) compiler that switches between interpreter and native compilation at runtime.

Native Image

GraalVM introduces ahead-of-time (AOT) where the language implementation such as JavaScript is compiled down into a native binary representation for fast processing.

Virtual Machine

The ecosystem comes with a fully memory managed VM with garbage collector and includes security features such as memory isolation and sandboxing. The virtual machine comes with development tools such as a live debugger.

Use Case Scenario:

Let's take an example where the user has a JavaScript application that sanitizes inputs and stores them inside the database. The sanitization and validation implementation would be dependent on the client application capability, such as which language it's written in, which sanitization package is being used, and with what version. Having multiple clients on different platforms is a common use case where data is being fed into a central back-end database server. Such a solution can lead to discrepancies in data quality in the database server.

Performing the data cleaning in the database server would be an ideal central location. However, simple validation rules are hard to implement as they require international standardization knowledge, and the problem becomes much worse if the

implementation needs to be done in a SQL dialect-based stored program. For example, if the requirement is to verify if the given string is a valid email address, social security number, ISSN, URL, mobile number, and postal code to name a few.

This is where procedural languages such as JavaScript become very useful. Not only do they make the development easy, but the user can easily reuse already available packages that specialize in making sure that such validation is based on standards.

Development Experience

Traditionally, database systems have their own procedural programming languages for stored programs. These languages are unfamiliar to most developers, and they often suffer from the lack of third-party libraries and poor support from development tools. Different language dialects also make it difficult to port routines between database systems. Compared to modern programming languages, program execution is also slower since it is based on interpreted code.

Defining JavaScript Stored Programs

To overcome the challenges discussed above, we are introducing support for JavaScript stored programs. To define a JavaScript stored program in MySQL, we use the same SQL statements as for traditional stored functions and procedures:

```
CREATE FUNCTION construct_url (path VARCHAR(50),
search VARCHAR(20)) RETURNS VARCHAR(100)
LANGUAGE JAVASCRIPT AS $$
  let url = `${path}
    ${search && !search.startsWith('?') ? '?' : ''}
    ${search ?? ''}`;
  return encodeURI(url);
$$
```

Note that a LANGUAGE clause is used to specify the language of the stored function. (Valid languages are SQL and JAVASCRIPT.) The JavaScript code is specified in a new AS clause as an ordinary character string enclosed in quotes.

```
CREATE FUNCTION construct_url (path VARCHAR(50),
search VARCHAR(20)) RETURNS VARCHAR(100)
LANGUAGE JAVASCRIPT AS $JS_CODE$
  let url = `${path}
    ${search && !search.startsWith('?') ? '?' : ''}
    ${search ?? ''}`;
  return encodeURI(url);
$JS_CODE$
```

Since the JavaScript code may itself contain single and double quotes, we are introducing a new quoting mechanism, called *dollar quotes*, that can be used to avoid conflicts. A dollar-quote is a character sequence consisting of a dollar sign ($), an optional "tag" of zero or more characters, and another dollar sign. Above is an example

of a function that uses a regular expression to check if a prefix of the string argument matches a suffix of the same string.

As seen from the above examples, the JavaScript code is embedded directly in the definition of the SQL-callable function. The names of the arguments can be referred directly in the JavaScript code, and when the function is called, there will be an implicit type conversion between SQL types and JavaScript types.

Executing JavaScript inside SQL statements

A JavaScript function may be called from SQL statements anywhere a traditional SQL function may be called; in SELECT expressions, WHERE, GROUP BY, and ORDER BY clauses, DMLs, DDLs, Views etc... Here is an example of an SQL statement that calls the string similarity function for each row of the table to find the Top-K most similar strings for a given reference string.

```
CREATE FUNCTION string_similarity(
  s1 VARCHAR(255), s2 VARCHAR(255)) RETURNS INT
LANGUAGE JAVASCRIPT AS $$
  const [ str1, str2, len1, len2 ] =
    s1.length < s2.length ?
      [ [...s2], [...s1], s2.length, s1.length ] :
      [ [...s1], [...s2], s1.length, s2.length ];
  var res = [...Array( len1 + 1 )].map( x =>
     [...Array(len2 + 1).keys()]
  );
  str1.forEach( (c,i) => {
    res[i+1][0] = i+1;
    str2.forEach( (d,j) => {
      res[i+1][j+1] = Math.min(
        res[i][j+1] + 1,
        res[i+1][j] + 1,
        res[i][j] + ((c == d) ? 0 : 1)
      )
    })
  })
  return res[len1][len2];
$$

-- Query: Top-K Most Similar Strings from Table
SELECT string_similarity (my_col, "reference string")
FROM my_table
ORDER BY string_similarity (my_col, "reference string")
LIMIT 25
```

To call JavaScript stored procedures, the CALL statement should be used. Both input and output parameters are supported for stored procedures. More on stored procedures below.

Executing SQL inside JavaScript code

Executing JavaScript stored functions inside SQL statements is one way to interact with MySQL data. The feature also provides an interface to issue SQL queries inside JavaScript, which is particularly useful for JavaScript stored procedures that cannot be invoked inside SELECT, DMLs, and DDL statements.

Both simple SQL statements and prepared statements are supported with full parameter binding support. In the example below we demonstrate using a simple SELECT query that iterates over constructed URLs built via the stored function in the prior example. The constructed URLs are then inserted into a table using prepared statement.

```
CREATE PROCEDURE store_urls (OUT url_inserted INT)
LANGUAGE JAVASCRIPT AS $$

  let selectQuery = mysql.getSession ().sql(
    `SELECT construct_url(path, product)
     FROM my_table`);

  let insertQuery = mysql.getSession ().prepare(
    `INSERT INTO my_urls(url) VALUES (?)`);

  url_inserted = 0;
  let result = selectQuery.execute(), row = null;
  while(row = result.fetchOne()) {
    insertQuery.bind(row[0]).execute();
    url_inserted++;
  }

$$
```

As for traditional SQL stored procedures, there are two modes for the user to consume query results. First, as shown above, the user can use a cursor mode where the query result can be iterated in JavaScript. The second option is a cursor-less mode where the query result set is returned directly to the caller. Furthermore, JavaScript stored procedures enable cursors with prepared statements, which is not possible currently with traditional SQL stored procedures.

The data access API used is the MySQL JavaScript XDevAPI already available in the Node.js MySQL connector and MySQL shell. The API ensures that the data type conversion is seamless both for query result set and bind parameters. It fully supports transactions, session variables, access to diagnostic information such as errors and warnings and result set metadata.

Debuggability

The "JavaScript for MySQL HeatWave" feature focuses on the development experience, it provides tools to help users debug their code. The global console JavaScript object offers several methods that will write to either a standard IO stream or an error IO stream. The feature introduces new SQL interfaces (UDFs) to enable developers to access the standard output and error streams. For example:

```
SELECT mle_session_state("stdout");
```

Similarly, in case of runtime errors, the user might want to access the full stack trace in addition to the error message to troubleshoot the issue. Such information is also exposed via the same UDF interface. i.e.,

```
SELECT mle_session_state("stack_trace");
```

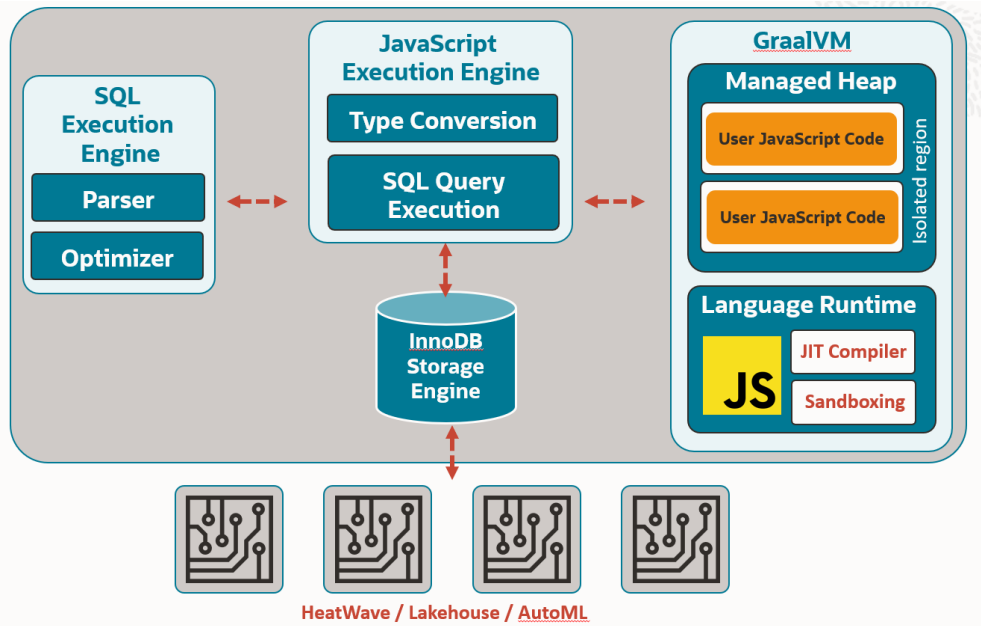
Cloud Ready Architecture

The new execution engine (GraalVM) is integrated inside the database server natively. The feature is designed from the ground-up with the cloud service as the primary focus. This provides smooth interaction with existing cloud features:

Compatibility: The feature is fully compatible with various server components such as enterprise thread pool, audit, and replication variations etc... Existing clients and connectors work seamlessly.

Storage engine: The feature is agnostic to storage engine, data can be accessed transparently from InnoDB, HeatWave, etc…

Interoperability: JavaScript stored programs work with traditional SQL stored programs. Interactions with features based on SQL stored programs, for example MySQL Autopilot and HeatWave AutoML, is seamless.



Resource Utilization

The feature is designed to be sensitive to resource utilization. More specifically, it uses resources effectively and hence has zero memory and compute impact on the server until a JavaScript stored program is invoked.

Memory Resources

The size of the GraalVM heap is automatically configured based on the size of the cloud instance. A larger heap comes with a larger shape provisioned. Moreover, the VM is fully managed and uses garbage collection to keep the peak memory footprint within configured limit. GraalVM also allows code sharing so that multiple stored programs instances do not occupy more space in the VM heap.

Compute Resources

JavaScript execution uses the same physical threads managed by the MySQL Enterprise Thread Pool (ETP). ETP allows for queuing the user operations and reusing a limited number of threads. Therefore, the JavaScript execution engine scales with the ETP configured limits for the cloud instance. Stored programs are also cached and reused in later re-execution for faster runtimes. Later executions will also be able to take advantage of any JIT compilation that may have been applied to the code by earlier executions. Hence, the more times the same code snippet is executed, the more efficient will the execution be.

Resource Observability

Users can now monitor the feature state and resource utilization using MySQL status variables.*'mle_heap_usage'* provides the % VM memory allocated while *'mle_heap_status'* allows the user to know if the VM is currently busy in garbage collection. In addition, the SQL interface metrics are also integrated with cloud monitoring tools.

Security

The "JavaScript for MySQL HeatWave" feature is part of the cloud offering and as such offers the highest levels of security, isolation, and data protection. As discussed below, "JavaScript for MySQL Heatwave" relies on the industry-proved Oracle's GraalVM security guarantees.

Resource Restriction

Using a custom-built VM with MySQL enables fine grained control over the sandboxing policy to restrict access from the VM and JavaScript user code.

Memory Isolation: The VM ensures that no memory beyond what's allocated will be used – malicious code cannot compromise other modules of the MySQL server. Every stored program is parsed and executed in its own context. This isolation policy does not allow for one stored program to read or modify other stored programs' data or code.

Compute Restrictions: Spawning or manipulating threads from JavaScript user code is restricted. Furthermore, access to 'evaluate' dynamic JavaScript source is restricted.

Since no new threads can be created in JavaScript code, the sandboxing ensures that malicious code cannot slow down the server excessively.

Network Restriction: The JavaScript user code has no access to sockets or network communication.

File System Restrictions: The VM has no file system footprint, and the JavaScript user code is not allowed to use the persistent storage directly.

Privileges

The feature builds on the standard MySQL privileges model. Only privileged users are allowed to create stored programs. The access to the SP can be controlled by other privileges. One user may define the stored programs that can be executed by others, and the proper user privileges will be applied for SQL execution inside stored procedures.

Advanced Mitigations

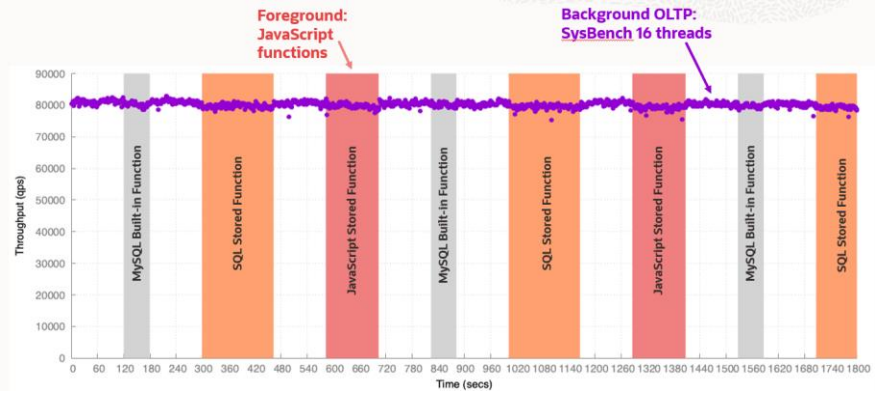
In addition to the above, the VM is configured to counter advanced security attacks such as JIT spraying attacks by using constant blinding and by using speculative optimization barriers preventing side-channel attacks such as Spectre and Meltdown.

Performance

Apart from reducing client-server data transfer latency, the integration support for JavaScript comes with additional performance benefits.

Native Integration

The JavaScript integration inside MySQL is done using a custom-built VM based on GraalVM's ahead-of-time (AOT) compilation technology. This allows the VM to be finely tuned to the MySQL requirements and cloud shape. The VM built includes custom memory management routines such as garbage collector and the Graal JIT compiler. The VM is natively integrated inside the database process which avoids any inter-process communication and serialization overheads. The VM is tightly integrated with the Enterprise thread pool where thread management and observability are executed by the database server. With this native integration, we achieve JavaScript execution without causing overhead on background OLTP tasks. Below we can see that the background OLTP is not impacted by JavaScript execution even though the underlying VM uses background tasks such as JIT compilation and garbage collection.



Graal.JS Implementation

The Graal compiler itself uses state-of-the-art optimizations such as aggressive inlining and partial escape analysis. On top of these optimizations, the VM uses profile guided JIT compilation where frequently executed JavaScript code will be compiled into native code to accelerate execution on the GraalVM. The resulting Graal JavaScript engine has comparable performance to the v8 JavaScript engine.

Conclusion

With the inclusion of the "JavaScript for MySQL HeatWave" feature (currently in limited availability), developers can now express complex programming logic directly inside the MySQL server. This allows developers to push data-intensive parts of their applications close to their data, reducing data movement. The use of JavaScript based on ECMAScript 2023 prevents vendor lock-in issues, while the user enjoys all the benefits of GraalVM (Enterprise Edition) at no additional cost. Furthermore, the feature integrates seamlessly with the MySQL HeatWave cloud service where the latest innovations are available to developers.

Resources

Learn more about MySQL HeatWave: https://www.oracle.com/mysql/

Learn more about GraalVM: https://www.oracle.com/java/graalvm/

Try MySQL HeatWave for free; https://www.oracle.com/mysql/free

References

[1] https://survey.stackoverflow.co/2023/#section-most-popular-technologies-programming-scripting-and-markup-languages

[2] https://www.graalvm.org/

[3] https://tc39.es/ecma262/2023/

[4] https://dev.mysql.com/doc/refman/8.0/en/sql-compound-statements.html

[5] https://w3techs.com/technologies/details/cp-javascript

[6] https://www.npmjs.com/

[7] https://nodejs.org/en

[8] https://deno.com/

[9] https://www.graalvm.org/javascript/

Connect with us

Call +**1.800.ORACLE1** or visit **oracle.com**. Outside North America, find your local office at: **oracle.com/contact**.

blogs.oracle.com facebook.com/oracle twitter.com/oracle