

ORACLE



Programmation stockée JavaScript dans MySQL Enterprise Edition

Webinar du 20 Mai 2025

Emmanuel COLUSSI

MySQL Solution Architect EMEA

Oracle MySQL GBU



Presentation Title

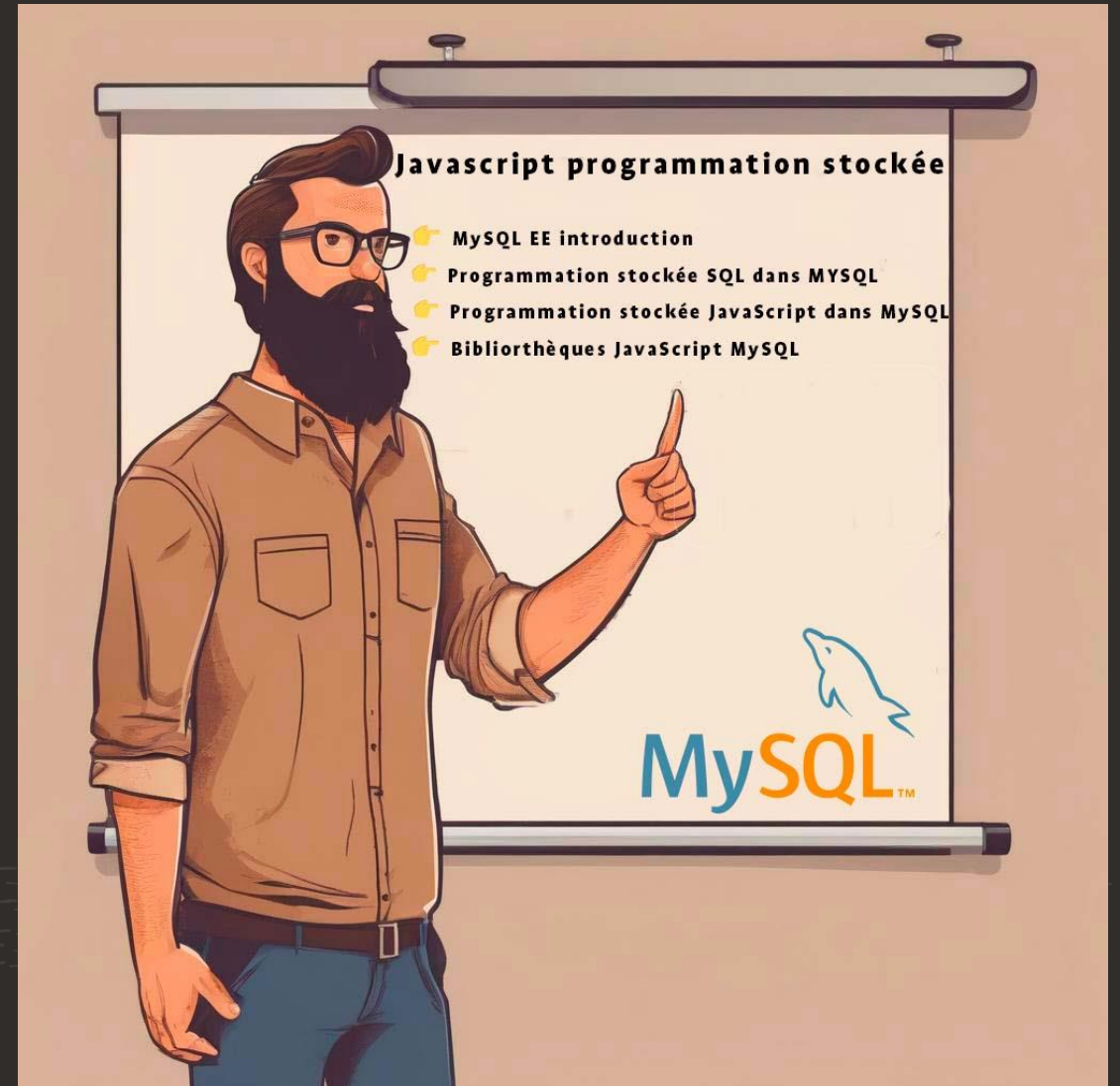
Speaker



Emmanuel COLUSSI
MySQL Solution Architect EMEA

Agenda

- 👉 Introduction à MySQL Enterprise Edition
- 👉 Programmation stockée SQL dans MySQL
- 👉 Programmation stockée JavaScript dans MySQL
- 👉 Bibliothèques JavaScript 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



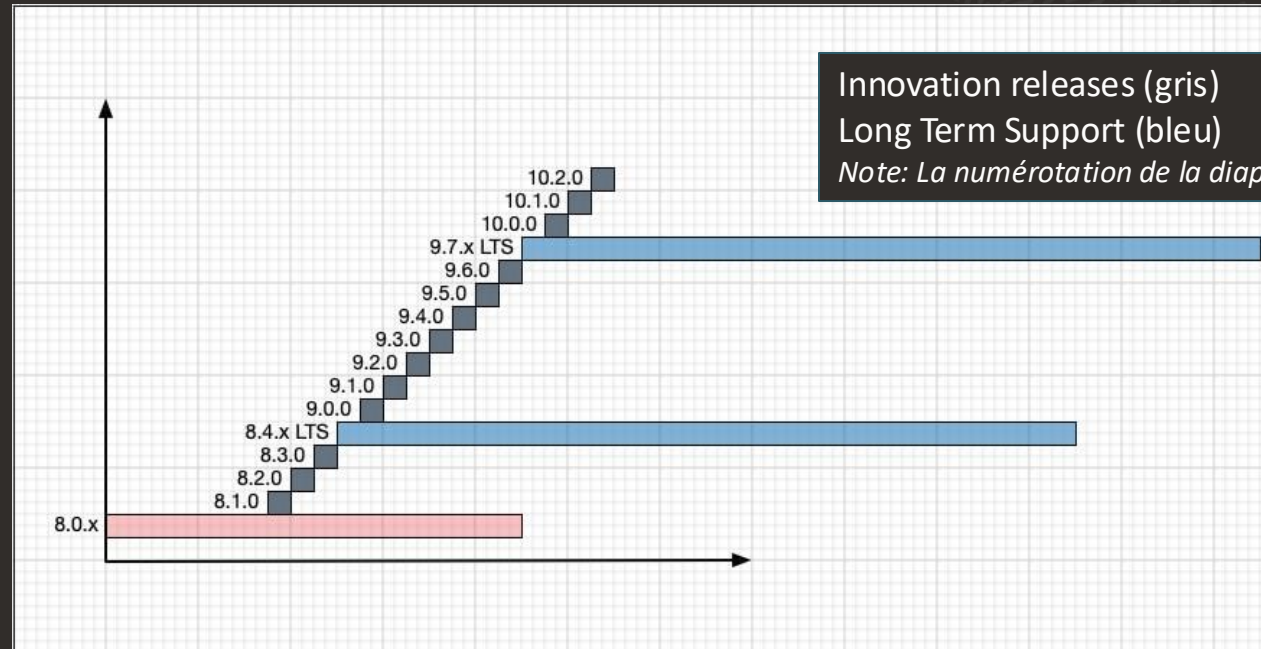
MySQL 24x7 Support

MySQL Community



Nouvelles versions : MySQL LTS & Innovation Releases

MySQL Community et Enterprise



Versions MySQL à support long terme (LTS)

- Correctifs de bugs et de sécurité uniquement
- Compatibilité ascendante
- Cycle de vie du support : 5 ans de support principal + 3 ans de support étendu
- Inclut des outils (comme MySQL Shell)

Versions MySQL Innovation

- Innovations de pointe
- Probablement publiées chaque trimestre
- Incluent des outils (comme MySQL Shell)
- Les connecteurs sont publiés uniquement en version Innovation

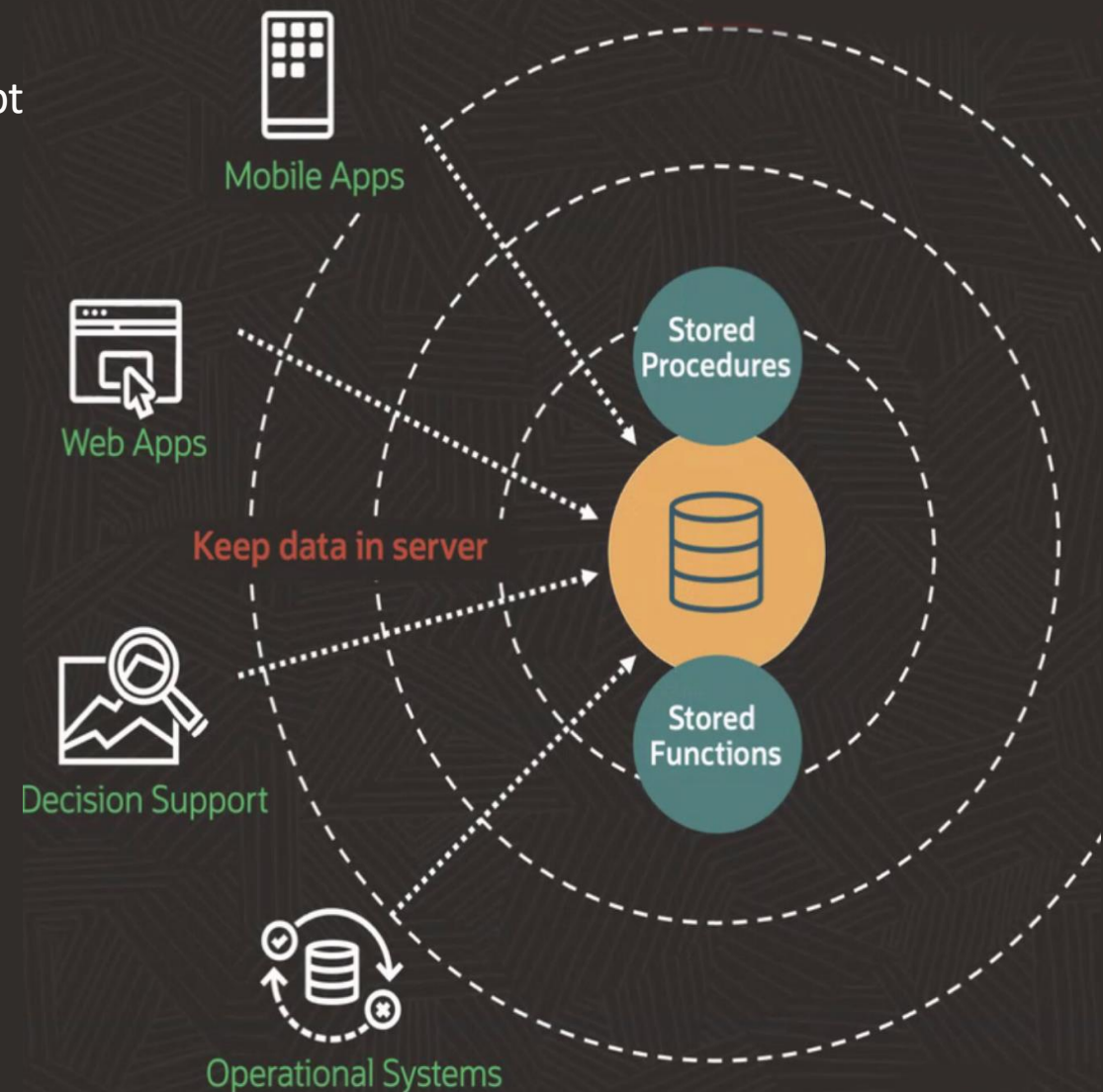
Programmes stockés dans MySQL Enterprise

Programmes stockés : réduisent les coûts, la complexité et améliorent la sécurité

Exécuter des procédures stockées et des fonctions stockées JavaScript
fonctions stockées via GraalVM

Gérer des fonctionnalités d'application à forte intensité de données
dans des programmes stockés.

- Réduire les coûts
- Améliorer la sécurité
- Simplifier les traitements lourds (ETL → ELT)
- Réduire les mouvements de données



Programmation stockée SQL dans MySQL



Routines stockées MySQL

Une routine stockée est un ensemble réutilisable d'instructions SQL ou JavaScript stockées sur le serveur

- Les clients invoquent la routine au lieu d'envoyer à chaque fois des instructions individuelles

Procédures stockées

- Créées avec CREATE PROCEDURE, invoquées avec CALL
- Ne renvoient pas de valeur, mais peuvent modifier des paramètres pour produire une sortie
- Peuvent générer des ensembles de résultats pour le client

Fonctions stockées

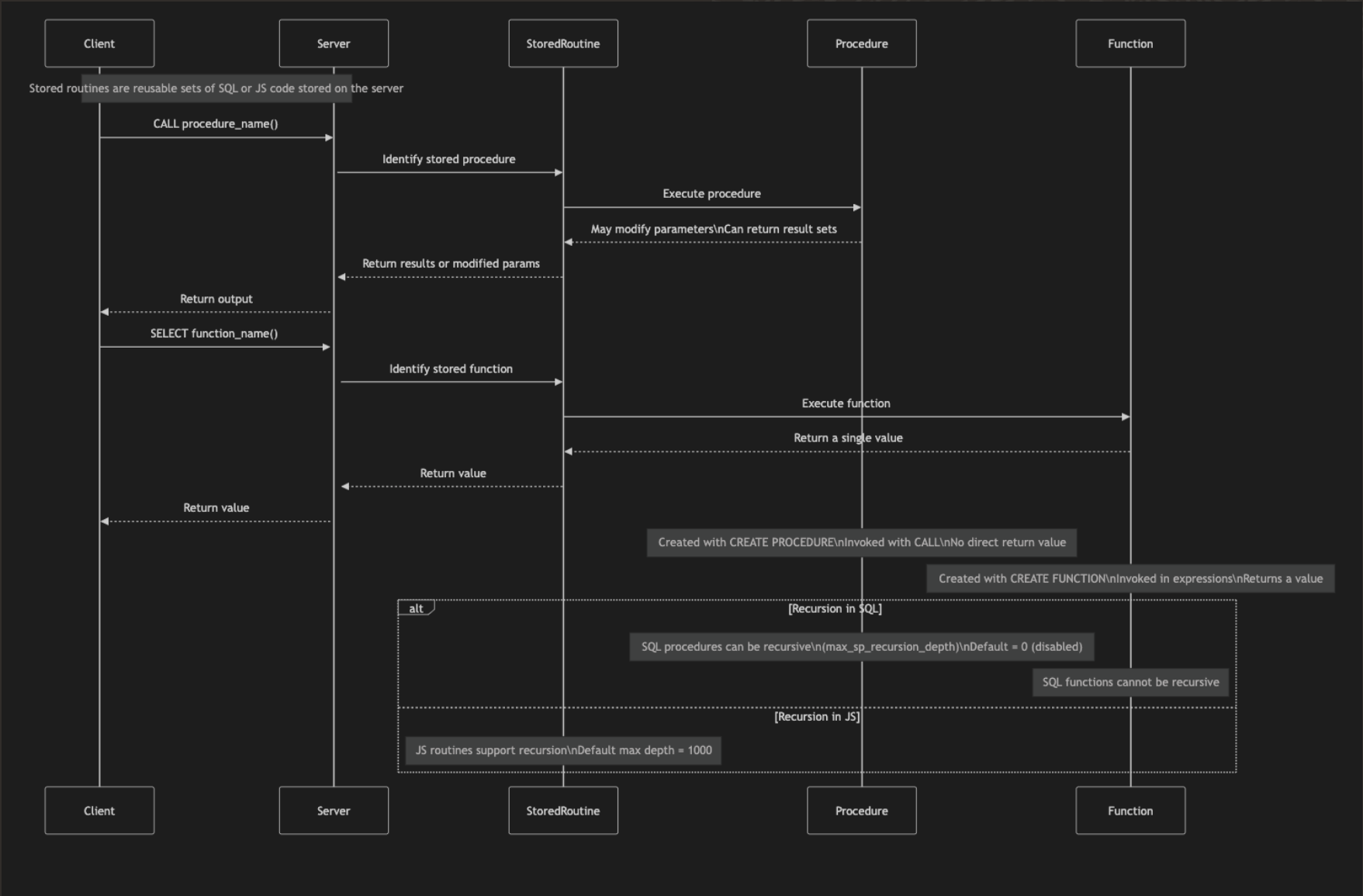
- Créées avec CREATE FUNCTION, utilisées comme des fonctions intégrées
- Invoquées dans des expressions et retournent une valeur
- Associées à une base de données spécifique
- Les fonctions SQL stockées ne peuvent pas être récursives

Règles de récursion

- Les procédures stockées SQL peuvent être récursives, mais la récursion est désactivée par défaut (max_sp_recursion_depth)
- Les routines stockées en JavaScript prennent en charge la récursion
- Profondeur maximale par défaut : 1000

Routines stockées MySQL

Flux



Sécurité des procédures stockées



Il existe des privilèges spécifiques pour les routines

- CREATE ROUTINE, ALTER ROUTINE, EXECUTE

SQL SECURITY définit le contexte de sécurité de l'objet stocké

- Si la définition omet l'attribut DEFINER, l'utilisateur qui crée l'objet en devient le propriétaire par défaut (definer)

SQL SECURITY DEFINER

- Exécute la routine dans le contexte de sécurité du créateur (definer)
- Permet à d'autres utilisateurs d'exécuter la routine même s'ils n'ont pas les privilèges sur les objets référencés

SQL SECURITY INVOKER

- Exécute la routine dans le contexte de sécurité de l'utilisateur appelant (invoker)
- Permet à d'autres utilisateurs d'exécuter la routine uniquement sur les objets pour lesquels ils ont les privilèges

Utilisations des routines stockées

Étendre les fonctions disponibles pour les clients

- Fonctionnalité centralisée côté client
- Créez une routine dans MySQL et rendez-la disponible pour plusieurs clients
- Maintenance simplifiée
- Sécurité renforcée

Créer des automatisations

- Utilisées dans des déclencheurs (triggers)
- Utilisées par le planificateur d'événements (event scheduler)

Sécurité

- Réduire l'accès aux données en exécutant avec les privilèges du DEFINER
- Traitement centralisé en un seul endroit



Exemple de procédure stockée SQL

```
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)

Exemple de fonction stockée SQL



```
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)

Métadonnées de routine stockées



Interroger la table ROUTINES dans la base INFORMATION_SCHEMA

- Utiliser SHOW CREATE PROCEDURE et SHOW CREATE FUNCTION pour afficher les définitions des routines
- Utiliser SHOW PROCEDURE STATUS et SHOW FUNCTION STATUS pour examiner les caractéristiques des routines

Jusqu'à MySQL 8.4 : les routines stockées sont uniquement en SQL

- Se référer au manuel pour les détails sur les variables et le contrôle de flux

À partir de MySQL 9.0 : prise en charge de JavaScript

Depuis MySQL 9.2 : possibilité de créer des bibliothèques pour améliorer la réutilisabilité des routines

Support JavaScript dans MySQL



Conformité à ECMAScript 2023

- Fonctionne en mode strict par défaut
- Prend en charge les objets intégrés standard: Object, Function, Math, Date, String, etc.

Environnement d'exécution étendu

- “global” et “globalThis” sont pris en charge mais limités à la routine actuelle
- Pas d'accès au système de fichiers ou au réseau à partir du code JavaScriptcode
- **Node.js n'est pas supporté**

Modèle d'exécution

- Monothread par requête
- Les fonctionnalités asynchrones (Promise, async/await) sont simulées — peuvent entraîner un comportement non déterminist

Interopérabilité avec SQL

- Appeler des programmes stockés SQL depuis du JavaScript
- Les routines JavaScript peuvent être invoquées depuis des procédures, fonctions, événements et déclencheurs SQL
- Prend en charge la récursion jusqu'à 1000 niveaux

Confort pour les développeurs

- Pas besoin de modifier le DELIMITER – le point-virgule (;) est géré de manière transparente

Conversion des types entre MySQL et JavaScript

- La plupart des types de données MySQL sont pris en charge. Pour les arguments d'entrée et de sortie des programmes stockés MLE, ainsi que pour les types de retour
 - Conversion automatique entre les types de données MySQL et JavaScript

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

- Détails ici: <https://dev.mysql.com/doc/refman/9.3/en/srjs-data-arguments.html>

Correspondance des types de données JavaScript vers MySQL – Points clés à considérer

- Typage Dynamique en JavaScript
 - JavaScript est un langage à typage dynamique ; les types de retour sont déterminés uniquement lors de l'exécution
 - Cela diffère du modèle de typage statique de SQL
- Conversion Automatique de Types
 - MySQL convertit automatiquement les types de retour de JavaScript en types MySQL compatibles
 - Consultez le Manuel MySQL 9.3 – [Conversion d'Arguments de Données](#) pour tous les détails
 - Exemples:
 - **JavaScript Boolean** → MySQL TINYINT, SMALLINT, CHAR, VARCHAR, FLOAT, etc.
 - **JavaScript String** → MySQL TINYINT, SMALLINT, CHAR, VARCHAR, FLOAT, etc.
- Notes Spéciales
 - Infinity et -Infinity sont traités comme des valeurs hors limite
 - NaN entraîne une erreur de conversion de type invalide
 - **Tout arrondi est effectué avec Math.round()**
 - **BigInt ou chaînes non numériques converties en FLOAT provoquent des erreurs de conversion**

Support du fuseau horaire en JavaScript

- Les programmes stockés en JavaScript utilisent le fuseau horaire de la session MySQL actif lors de leur première invocation
 - Ce fuseau horaire reste fixe pendant toute la durée de la session
- Modifier le fuseau horaire de la session ultérieurement n'affecte pas les programmes stockés déjà utilisés (en cache)
- Pour appliquer le nouveau fuseau horaire:
 - Call `mls_session_reset()` pour vider le cache
 - Les invocations suivantes utiliseront le fuseau horaire de la session mis à jour



Surcharge JavaScript des arguments SQL

- Ne pas redéclarer les arguments du programme en utilisant let, var ou const dans les programmes stockés en JavaScript
- La redéclaration des paramètres crée des variables locales qui masquent les arguments d'origine
- Cela rend les valeurs passées inaccessibles
- Utilisez toujours l'argument tel quel pour préserver sa valeur et son comportement d'origine

- 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)'
```

Exemple SQL/JavaScript : Plus Grand Diviseur Commun

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
*/
```

API SQL depuis JavaScript

- Objets supportés:
 - **Column** – Métadonnées pour les colonnes du jeu de résultats
 - **Row** – Représente une seule ligne dans un jeu de résultats
 - **PreparedStatement** – Gère l'exécution des instructions SQL préparées
 - **SqlExecute** – Exécute du SQL simple via execute()
 - **SqlResult** – Jeu de résultats renvoyé par une instruction SQL
 - **Warning** – Capture les avertissements générés lors de l'exécution de la instruction
 - **Session** – Représente la session utilisateur MySQL
 - Supporte startTransaction(), commit(), rollback()
 - Donne accès aux variables de session MySQL
- Fonctionnalités supplémentaires:
 - getFunction() / getProcedure() renvoient des objets Fonction JavaScript pour les routines stockées MySQL
 - Accès aux objets schéma, table, ligne, et colonne
 - L'espace de noms MySQL inclut:
 - Fonctions intégrées SQL
 - `SqlError` (équivalent à MySQL SIGNAL)
- Note: L'API SQL est disponible uniquement dans les procédures stockées JavaScript (non supportée dans les fonctions stockées JavaScript)

Exemple SQL dans 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
...
```


Informations sur la session JavaScript

- Si elle n'est pas effacée, la sortie de la console est ajoutée à l'état de la session existant à chaque exécution
- Le composant MLE fournit plusieurs fonctions chargeables pour travailler avec les sessions utilisateur MLE
- `mle_session_reset()`
 - Efface l'état actuel de la session MLE
 - Peut réinitialiser toutes ou certaines parties: "stdout", "stderr", "output"
- `mle_session_state()`
 - Renvoie des informations sur la session concernant le programme stocké MLE le plus récemment exécuté
 - Inclut la sortie accumulée de `console.log()` / `console.error()`
- `mle_set_session_state()`
 - Définit des règles de conversion pour faire correspondre les types MySQL INTEGER et DECIMAL aux valeurs JavaScript

Exemple

```
mysql> CREATE PROCEDURE helloworld_js()  
LANGUAGE JAVASCRIPT  
AS $mle$  
    console.log("Hello World")  
$mle$;  
  
mysql> CALL helloworld_js;  
Query OK, 0 rows affected (0.0072 sec)  
  
mysql> SELECT mle_session_state("stdout")  
AS 'STDOUT';  
+-----+  
| STDOUT |  
+-----+  
| Hello World |  
|  
+-----+
```

SqlError Object

- Vous pouvez créer une `SqlError` en utilisant le constructeur illustré ici:

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

- Lorsqu'une `SqlError` est levée, une erreur est générée dans MySQL, similaire à celle levée par une instruction `SIGNAL`, comme par exemple :

```
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`
```

Gestion des erreurs en JavaScript

- Une instruction SQL qui provoque des erreurs non gérées dans le programme stocké les renvoie au client
- Les erreurs peuvent être gérées à l'aide d'un ou plusieurs blocs **"try ... catch"**

```
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
```

Bibliothèques JavaScript dans MySQL

Bibliothèques JavaScript

- L'un des avantages de JavaScript est la possibilité de créer et de réutiliser des bibliothèques
 - Réduire la duplication de code
 - Mieux assurer la maintenance du code
 - Mieux encapsuler le code
 - Améliorer les performances
- La prise en charge du langage JavaScript dans MySQL est conforme à la norme ECMAScript 2023
- Les bibliothèques peuvent être :
 - Créées à l'intérieur de MySQL, avec une syntaxe spécifique
 - Inclure d'autres bibliothèques disponibles dans la même instance MySQL
 - Chargées à partir d'un fichier, en ajoutant un en-tête et un pied de page spécifiques à SQL
- Pour créer une bibliothèque, utilisez la déclaration `CREATE LIBRARY`, comme :

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


Importer des bibliothèques dans un programme stocké

- Une nouvelle clause USING a été introduite pour les fonctions et procédures stockées

- Exemple

```
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)
$mle$;
```

- La syntaxe JavaScript est vérifiée lors de la création de la bibliothèque, comme montré ici:

```
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
                ^
```

Import de bibliothèques externes

- Les bibliothèques externes, écrites selon la norme ECMAScript 2023, peuvent être importées en entourant le contenu de la bibliothèque entre les instructions de création de bibliothèque, par exemple

1. Renommez le fichier `my_library.js` en `my_library.sql`
2. Modifiez le fichier `mysqlibray.sql` en l'enfermant entre des instructions de création de bibliothèque

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

3. Chargez le fichier `mysqlibray.sql` comme d'habitude
`mysql -uroot -p < my_library.sql`
- Actuellement, il n'y a pas de résolution automatique des dépendances, donc vous devez encoder toutes les dépendances dans la même bibliothèque.
 - Utilisez Webpack, Rollup ou un outil similaire pour créer un bundle incluant tout

La puissance des bibliothèques : exemples

- Élargir les calculs mathématiques pour inclure des fonctions supplémentaires ou des types de données mathématiques, par exemple :
 - Math avancé pour la statistique et les calculs complexes
 - Vectors pour la genAI
- Ajouter des algorithmes d'apprentissage automatique
 - Pour extraire de nouvelles valeurs à partir des données existantes dans MySQL, sans exportation/importation
- Utiliser un plugin de validation pour empêcher l'enregistrement de valeurs invalides dans la base de données, pour des applications plus sécurisées
- Utiliser une bibliothèque de date/heure pour gérer des expressions comme "demain" ou "vendredi dernier", ou pour calculer la semaine calendaire (CW) d'une date et rendre les applications plus conviviales pour les utilisateurs et développeurs
- Manipuler les URI pour supprimer les mots de passe ou diviser les URI en utilisateur, hôte, paramètres, puis stocker séparément dans des colonnes, simplifiant ainsi le travail des développeurs
 - Et les colonnes générées peuvent exposer ces éléments directement dans une table
- Et bien plus encore...



Les points clés à retenir

Les points clés à retenir



	Programmes stockés SQL	JavaScript Stored Programs
Expressivité	❌ Difficile à utiliser, manque de constructions de base	✅ Très expressif et robuste
Efficacité	❌ Difficulté d'optimisation en raison du code interprété	✅ Les applications JavaScript sont rapides et optimisées par GraalVM
Ecosystème	❌ Manque de soutien de la part des IDE, des débogueurs, des cadres de test, etc.	✅ Un écosystème massif d'outils pour les développeurs d'applications JavaScript
Développeurs	❌ Manque de programmeurs expérimentés au sein de l'écosystème	✅ JavaScript est le langage de développement le plus populaire
Bibliothèques tierces réutilisables	❌ Peu d'exemples, principalement des exemples de code	✅ Millions

Resources

- Programmation stockée MySQL
 - <https://dev.mysql.com/doc/refman/9.3/en/stored-objects.html>
- JavaScript dans MySQL
 - <https://dev.mysql.com/doc/refman/9.3/en/stored-routines-js.html>
- Exemples JavaScript MySQL tirés du manuel
 - <https://dev.mysql.com/doc/refman/9.3/en/srjs-examples.html>
- Exemple JavaScript MySQL : opérations vectorielles dans MySQL
 - <https://lefred.be/content/mysql-vector-datatype-create-your-operations-part-1/>
- Exemple JavaScript MySQL : UUID
 - <https://blogs.oracle.com/mysql/post/javascript-support-in-mysql-the-uuid-example>

Merci

Emmanuel COLUSSI

emmanuel.colussi@oracle.com



ORACLE

