MySQL
User-Defined Functions

...in JavaScript!

https://github.com/rpbouman/mysqlv8udfs
Welcome!

- @rolandbouman
- roland.bouman@gmail.com
- http://rpbouman.blogspot.com/
- http://www.linkedin.com/in/rpbouman
- http://www.slideshare.net/rpbouman
- Ex-MySQL AB, Ex-Sun Microsystems
- Currently at http://www.pentaho.com/

https://github.com/rpbouman/mysqlv8udfs
MySQL Programmability

- SQL
- Persistent Stored Modules (Stored Routines)
- User-defined functions (UDFs)
MySQLv8UDFs: JavaScript Programmability

- https://github.com/rpbouman/mysqlv8udfs
- Based on Google's V8
- More than just executing JavaScript:
  - Scriptable front for MySQL's native UDF interface
MySQL stored routines

• “Standard” SQL/PSM syntax
  – Scalar functions
  – Procedures
  – Triggers
  – Events
• Stored in the data dictionary
• Interpreted

https://github.com/rpbouman/mysqtv8udfs
MySQL UDFs

- External binary library (typically C/C++):
  - Scalar functions
  - Aggregate functions
- Registered in the data dictionary
- Compiled Native code
JavaScript UDFs. Why?

- Started as a UDF example
- Inspired by drizzle's `js()` function
- Turned out to have real benefits:
  - Convenient manipulation of JSON blobs
  - Safer and easier than 'real' C/C++ UDFs
  - More expressive than SQL/PSM
  - Sometimes much faster than stored routines*
Intermezzo: Easter day Performance comparison

Easter Day Performance (1.000.000)

- SQL Expression: 4.61 seconds
- SQL Stored Function: 12.53 seconds
- JavaScript UDF: 2.66 seconds

https://github.com/rpbouman/mysqlv8udfs
The mysqlv8udfs project

- Scalar Functions:
  - js()
  - jsudf()
  - jserr()

- Aggregate Functions:
  - jsagg()

- Daemon plugin*:
  - JS_DAEMON

https://github.com/rpbouman/mysqlv8udfs
The js() UDF

- `js(script[, arg1, ..., argN])`
  - Execute script
  - Return value (as string) of the last js expression

- Optional arguments `arg1 ... argN`
  - Accessible via the built-in `arguments` array
  - `arg1` accessible as `arguments[0]` (and so on)

- Script*
  - if constant it is compiled only once
  - executed for each row

https://github.com/rpbouman/mysqlv8udfs
The MySQL UDF interface

• Simple functions (scalars) and aggregates
• Native library callbacks (calling sequence)
  – Plugin directory
  – `mysql.func` table
• Data structures
  – Return value: `struct UDF_INIT *initid`
  – Arguments: `struct UDF_ARGS *args`

```sql
CREATE [AGGREGATE] FUNCTION name
RETURNS (STRING | REAL | INTEGER)
SONAME 'libraryfile'
```

https://github.com/rpbouman/mysqlv8udfs
UDF JavaScript binding

- Scalars: jsudf(), Aggregates: jsagg()
  - Return value is always a MySQL STRING
- Script argument:
  - Constant. Compiled and immediately executed (1x)
  - JavaScript callbacks defined in script are called during the native UDF calling sequence
- UDF data structures scriptable at runtime:
  - Members of struct UDF_INIT appear as js globals
  - struct UDF_ARGS as global arguments object

https://github.com/rpbouman/mysqlv8udfs
The jsudf() UDF

- jsudf(script[, arg1, ..., argN])
  - Call the init() callback (optional)
  - For each row, return the result of the udf() callback
  - Call the deinit() callback (optional)
jsudf() Argument processing

• Arguments beyond the initial script argument:
  - Values passed to the udf() callback
  - UDF_ARGS scriptable as global arguments array
  - WARNING: In javascript functions, the local built-in arguments object refers to actual arguments
  - Local arguments mask the global arguments object.
  - Use this.arguments to refer to the global array of argument objects.

• Use init() to validate or pre-process arguments

https://github.com/rpbouman/mysqlv8udfs
The Argument object

- **name**: Expression text. If provided, the alias
- **type**: code indicating the runtime data type
  - 0: STRING_RESULT, 1: REAL_RESULT, 4: DECIMAL_RESULT
- **max_length**: maximum string length
- **maybe_null**: true if nullable
- **const_item**: true if value is constant
- **value**: argument value
<table>
<thead>
<tr>
<th>Type family</th>
<th>MySQL column data type</th>
<th>MYSQL UDF data type</th>
<th>v8 type</th>
<th>JS Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integral numbers</td>
<td>BIGINT</td>
<td>INT_RESULT</td>
<td>v8::Integer or v8::Number</td>
<td>Number</td>
</tr>
<tr>
<td></td>
<td>INT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MEDIUMINT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SMALLINT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TINYINT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Floating point numbers</td>
<td>DOUBLE</td>
<td>REAL_RESULT</td>
<td>v8::Number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FLOAT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decimal numbers</td>
<td>DECIMAL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary String</td>
<td>BINARY</td>
<td></td>
<td>v8::String</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>BLOB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LONGBLOB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MEDIUMBLOB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARBINARY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TINYBLOB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Character String</td>
<td>CHAR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LONGTEXT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MEDIUMTEXT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARCHAR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TEXT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TINYTEXT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structured String</td>
<td>ENUM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SET</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Temporal</td>
<td>DATE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DATETIME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TIME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TIMESTAMP</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The jsagg() UDF

- jsagg(script[, arg1, ..., argN])
  - Call the init() callback (optional)
  - Calls clear() before processing a group of rows
  - For each row in a group, the udf() callback is called
  - After processing a group, the agg() is called to return the aggregate value
  - Call the deinit() callback (optional)
The `jsagg()` UDF

```
init() -> clear() -> udf() -> agg() -> More groups? (Yes/No)
```

- More rows? (Yes/No)
- deinit() (No/Yes)

[GitHub Link](https://github.com/rpbouman/mysqlv8udfs)
JavaScript Environment

• JavaScript Standard built-ins:
  – Constructors (Date, RegExp, String etc.)
  – Static objects (JSON, Math)
  – Misc. functions (decodeURI, parseInt etc.)

• Globals provided by mysqlv8udfs *
  – arguments[] array
  – Some UDF interface variables and constants
  – require() function
  – console object
  – mysql object

https://github.com/rpbouman/mysqlv8udfs
The `require()` function

- Inspired by commonjs Module loading
- **Signature:** `require(filename[, , reload])`
  - Loads script file from the `js_daemon_module_path`
  -Executes the script and returns the result
  -Script is compiled and cached for reuse
  -Pass `true` as 2nd argument to force reload from file
- **`js_daemon_module_path`**
  -Read-only system variable of the JS_DAEMON plugin
  -Specified at `mysqld` command line or option file
  -Prevent loading arbitrary script files

https://github.com/rpbouman/mysqlv8udfs
The console object

• Inspired by console object in web-browsers

• Methods:
  - `log([arg1, ..., argN])`
  - `info([arg1, ..., argN])`
  - `error([arg1, ..., argN])`
  - `warn([arg1, ..., argN])`

• Write arguments to a line on the standard error stream
  - Typically ends up in the mysql error log

• `info()`, `error()`, and `warn()` include a header:
  - `2013-09-17 00:50:22 JS_DAEMON [info]: ...`
The `mysql` object

- Namespace for interacting with MySQL
  - Depends on libmysqlclient

```plaintext
mysql
  client
    connect()
  version

connection
  close()
  commit()
  rollback()
  query()
  setAutocommit()

query
  execute()
  result()
  done
  sql

resultinfo
  done: true
  rowCount
  type: "resultinfo"

resultset
  buffered
  done
  fieldCount
  type: "resultset"
  field()
  row()
```

0: "decimal"
1: "tinyint"
... ...
255: "geometry"

https://github.com/rpbouman/mysqlv8udfs
Oracle JSON functions (Labs)

- JSON, not JavaScript
  - Stick with this if you only need JSON manipulation
- That said....
  - Mysqlv8udfs performance is typically slightly better, in some cases substantially better
  - JSON functions have a few bugs which can be easily worked around using mysqlv8udfs
  - Easy to emulate with mysqlv8udfs, see sample implementations in the js project dir

https://github.com/rpbouman/mysqlv8udfs
JSON functions vs MySQLv8UDFs

https://github.com/rpbouman/mysqlv8udfs
Finally...

- Fork it on github. I appreciate your interest!
  - https://github.com/rpbouman/mysqlv8udfs
  - https://github.com/rpbouman/mysqlv8udfs/wiki
Questions?

https://github.com/rpbouman/mysqlv8udfs