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/

MySQL Programmability

- Persistent Stored Modules (Stored Routines)
- User-defined functions (UDFs)

MySQL stored routines

- “Standard” SQL/PSM syntax
- Scalar functions, procedures, triggers
- Stored in the data dictionary
- Interpreted

MySQL UDFs

- External binary library (typically C/C++)
- Scalar and aggregate functions
- Registered in the data dictionary
- Compiled Native code

https://github.com/rpbouman/mysqlv8udfs

UDFs to execute JavaScript

- https://github.com/rpbouman/mysqlv8udfs
- Based on Google's V8

JavaScript UDFs. Why?

- Started as an non-trivial UDF example
- Kinda like drizzle's `js()` function
- Turned out to have real benefits:
  - Convenient manipulating of JSON blobs
  - Safer and easier than 'real' C/C++ UDFs
  - More expressive than SQL/PSM
  - Sometimes much faster than stored routines*

https://github.com/rpbouman/mysqlv8udfs

Intermezzo: Easter day as stored SQL function

```
CREATE FUNCTION easter_day(dt DATETIME) RETURNS DATE
DETERMINISTIC NO SQL SQL SECURITY INVOKER
COMMENT 'Returns date of easter day for given year'
BEGIN
    DECLARE p_year SMALLINT DEFAULT YEAR(dt);
    DECLARE a      SMALLINT DEFAULT p_year % 19;
    DECLARE b      SMALLINT DEFAULT p_year DIV 100;
    DECLARE c      SMALLINT DEFAULT p_year % 100;
    DECLARE e      SMALLINT DEFAULT b % 4;
    DECLARE h      SMALLINT DEFAULT (19*a + b - (b DIV 4) - (
                                      (b - ((b + 8) DIV 25) + 1) DIV 3
                                    ) + 15) % 30;
    DECLARE L      SMALLINT DEFAULT (32 + 2*e + 2*(c DIV 4) - h - (c % 4)) % 7;
    DECLARE v100   SMALLINT DEFAULT h + L - 7*((a + 11*h + 22*L) DIV 451) + 114;

    RETURN STR_TO_DATE(
                CONCAT(
                    p_year
                ,   '-'
                ,    v100 DIV 31
                ,   '-'
                ,   (v100 % 31) + 1
                )
            ,   '%Y-%c-%e'
            );
END;
```

Intermezzo: Easter day in JavaScript (js UDF)

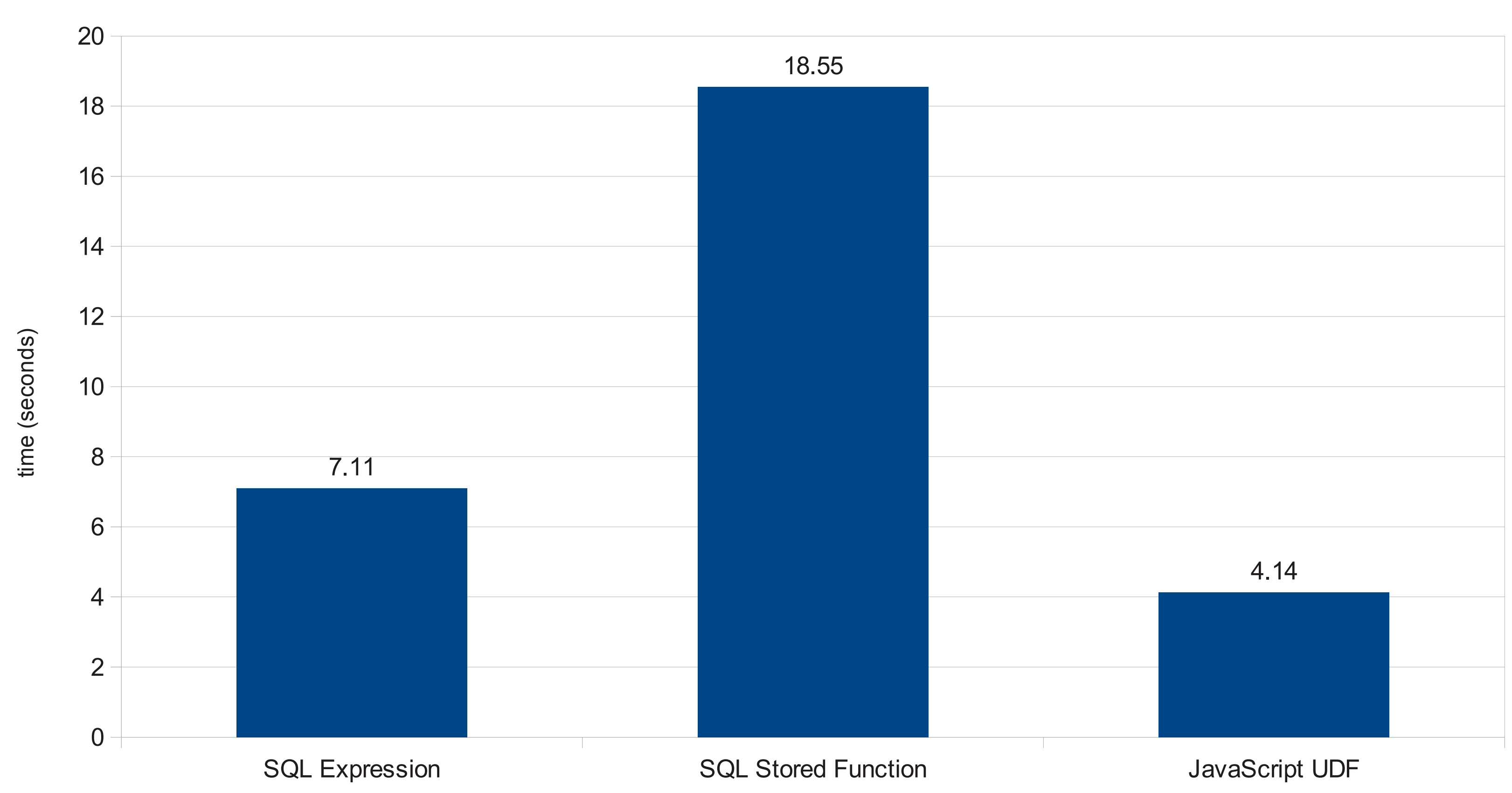
```
mysql> SELECT js('
    '> var y = parseInt(arguments[0].substr(0,4), 10),
    '>     a = y % 19, b = Math.floor(y / 100),
    '>     c = y % 100, d = Math.floor(b / 4),
    '>     e = b % 4, f = Math.floor((b + 8) / 25),
    '>     g = Math.floor((b - f + 1) / 3),
    '>     h = (19 * a + b - d - g + 15) % 30,
    '>     i = Math.floor(c / 4), k = c % 4,
    '>     L = (32 + 2 * e + 2 * i - h - k) % 7,
    '>     m = Math.floor((a + 11 * h + 22 * L) / 451),
    '>     n = h + L - 7 * m + 114,
    '>     M = Math.floor(n/31), D = (n%31)+1;
    '>     if (M < 10) M = "0" + M;
    '>     if (D < 10) D = "0" + D;
    '>
    '>     y + "-" + M + "-" + D;
    '>
    '>', NOW());
```

https://github.com/rpbouman/mysqlv8udfs

Intermezzo: Easter day as SQL expression

```
STR_TO_DATE(CONCAT(YEAR(now()), '-', (((19*(YEAR(now()) % 19) + (YEAR(now()) DIV
100) - ((YEAR(now()) DIV 100) DIV 4) - (((YEAR(now()) DIV 100) - (((YEAR(now())
DIV 100) + 8) DIV 25) + 1) DIV 3) + 15) % 30) + ((32 + 2*((YEAR(now()) DIV 100) %
4) + 2*((YEAR(now()) % 100) DIV 4) - ((19*(YEAR(now()) % 19) + (YEAR(now()) DIV
100) - ((YEAR(now()) DIV 100) DIV 4) - (((YEAR(now()) DIV 100) - (((YEAR(now())
DIV 100) + 8) DIV 25) + 1) DIV 3) + 15) % 30) - ((YEAR(now()) % 100) % 4)) % 7) -
7*(((YEAR(now()) % 19) + 11*((19*(YEAR(now()) % 19) + (YEAR(now()) DIV 100) -
((YEAR(now()) DIV 100) DIV 4) - (((YEAR(now()) DIV 100) - (((YEAR(now()) DIV 100)
+ 8) DIV 25) + 1) DIV 3) + 15) % 30) + 22*((32 + 2*((YEAR(now()) DIV 100) % 4) +
2*((YEAR(now()) % 100) DIV 4) - ((19*(YEAR(now()) % 19) + (YEAR(now()) DIV 100) -
((YEAR(now()) DIV 100) DIV 4) - (((YEAR(now()) DIV 100) - (((YEAR(now()) DIV 100)
+ 8) DIV 25) + 1) DIV 3) + 15) % 30) - ((YEAR(now()) % 100) % 4)) % 7)) DIV 451) +
114) DIV 31, '-', ((((19*(YEAR(now()) % 19) + (YEAR(now()) DIV 100) -
((YEAR(now()) DIV 100) DIV 4) - (((YEAR(now()) DIV 100) - (((YEAR(now()) DIV 100)
+ 8) DIV 25) + 1) DIV 3) + 15) % 30) + ((32 + 2*((YEAR(now()) DIV 100) % 4) +
2*((YEAR(now()) % 100) DIV 4) - ((19*(YEAR(now()) % 19) + (YEAR(now()) DIV 100) -
((YEAR(now()) DIV 100) DIV 4) - (((YEAR(now()) DIV 100) - (((YEAR(now()) DIV 100)
+ 8) DIV 25) + 1) DIV 3) + 15) % 30) - ((YEAR(now()) % 100) % 4)) % 7) -
7*(((YEAR(now()) % 19) + 11*((19*(YEAR(now()) % 19) + (YEAR(now()) DIV 100) -
((YEAR(now()) DIV 100) DIV 4) - (((YEAR(now()) DIV 100) - (((YEAR(now()) DIV 100)
+ 8) DIV 25) + 1) DIV 3) + 15) % 30) + 22*((32 + 2*((YEAR(now()) DIV 100) % 4) +
2*((YEAR(now()) % 100) DIV 4) - ((19*(YEAR(now()) % 19) + (YEAR(now()) DIV 100) -
((YEAR(now()) DIV 100) DIV 4) - (((YEAR(now()) DIV 100) - (((YEAR(now()) DIV 100)
+ 8) DIV 25) + 1) DIV 3) + 15) % 30) - ((YEAR(now()) % 100) % 4)) % 7)) DIV 451) +
114) % 31) + 1), '%Y-%c-%e')
```

Intermezzo: Easter day Performance comparison



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_DAEMON Plugin

```
mysql> SHOW VARIABLES LIKE 'js%';
+-----------------------+--------------------------------------+
| Variable_name         | Value                                |
+-----------------------+--------------------------------------+
| js_daemon_module_path | /home/rbouman/mysql/mysql/lib/plugin |
+-----------------------+--------------------------------------+
1 row in set (0.03 sec)

mysql> SHOW STATUS LIKE 'js%';
+-----------------------------------+------------+
| Variable_name                     | Value      |
+-----------------------------------+------------+
| js_daemon_version                 | 0.0.1      |
| js_v8_heap_size_limit             | 2048       |
| js_v8_heap_size_total             | 942944256  |
| js_v8_heap_size_total_executable  | 959591424  |
| js_v8_heap_size_used              | 892941672  |
| js_v8_is_dead                     | false      |
| js_v8_is_execution_terminating    | false      |
| js_v8_is_profiler_paused          | true       |
| js_v8_version                     | 3.7.12.22  |
+-----------------------------------+------------+
9 rows in set (0.00 sec)
```

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 `js()` UDF: Example

```
mysql> SELECT js('
    '>   arguments[0] + arguments[1];
    '> ', 1, 2) AS example
    -> ;
+----------+
| example |
+----------+
| 3        |
+----------+
1 row in set (0.03 sec)
```

https://github.com/rpbouman/mysqlv8udfs

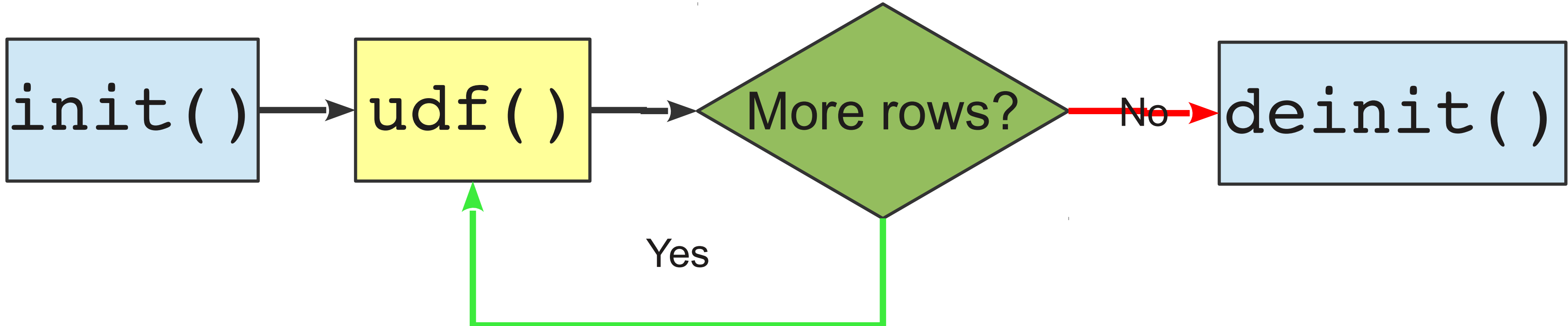
Binding the UDF interface to JavaScript

- Two UDFs:
  - `jsudf()` - scalar
  - `jsagg()` - aggregate
- Script argument:
  - Must be a constant string.
  - Compiled and immediately executed (once)
  - JavaScript callbacks defined in the script called during various stages in the UDF calling sequence
- UDF data structures scriptable at runtime

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)



https://github.com/rpbouman/mysqlv8udfs

`jsudf()` example: running total

```
mysql> SELECT amount, jsudf('
    ->   var total;
    ->   function init(){
    ->     console.info("Init");
    ->     total = 0;
    ->   }
    ->   function udf(num){
    ->     console.info("processing row");
    ->     return total += num;
    ->   }
    ->   function deinit(){
    ->     console.info("Deinit");
    ->   }
    -> ', amount) AS running_total
    -> FROM sakila.payment ORDER BY payment_date
```

`jsudf()` example: resultset and error log

```
+--------+---------------------+
| amount | running_total       |
+--------+---------------------+
|   2.99 | 2.99                |
|   2.99 | 5.98                |
.    ...  .  ...               .
|   4.99 | 67416.509999921     |
+--------+---------------------+
16049 rows in set (0.29 sec)
```

```
2013-09-16 14:31:44 JS_DAEMON [info]: Init
2013-09-16 14:31:44 JS_DAEMON [info]: processing row
.... .. .. .. .. .. .. ...... ......  ..............
2013-09-16 14:31:44 JS_DAEMON [info]: processing row
2013-09-16 14:31:44 JS_DAEMON [info]: Deinit
```

`jsudf()` Argument processing

- Arguments beyond the initial script argument:
  - Values passed to the **`udf()`** callback
  - argument objects available in global **`arguments`** array
  - WARNING: Inside functions, **`arguments`** refers to the arguments of the function (masking the global arguments object). Use **`this.arguments`** to refer to the global array of argument objects.
- Argument object describes argument (metadata)
- Use **`init()`** to validate or pre-process arguments

`jsudf()` arguments


```
[
 {
  "name": "'string'",
  "type": 0,
  "max_length": 6,
  "maybe_null": false,
  "const_item": true,
  "value": "string"
 },
 {
  "name": "real",
  "type": 1,
  "max_length": 8,
  "maybe_null": false,
  "const_item": true,
  "value": 3.141592653589793
 },
 {
  "name": "1",
  "type": 2,
  "max_length": 1,
  "maybe_null": false,
  "const_item": true,
  "value": 1
 },
 {
  "name": "2.3",
  "type": 4,
  "max_length": 3,
  "maybe_null": false,
  "const_item": true,
  "value": 2.3
 }
]
```

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

https://github.com/rpbouman/mysqlv8udfs

Argument Processing: Validating count and types

```
function init(){
  var args = this.arguments,
      nargs = args.length
  ;

  //validate the number of arguments:
  if (nargs != 1) throw "Expected exactly 1 argument";

  //validate argument type:
  var arg = args[0];
  switch (arg.type) {
    case REAL_RESULT:
    case INT_RESULT:
    case DECIMAL_RESULT:
      break;
    default:
      throw "Argument must be numeric";
  }
}
```

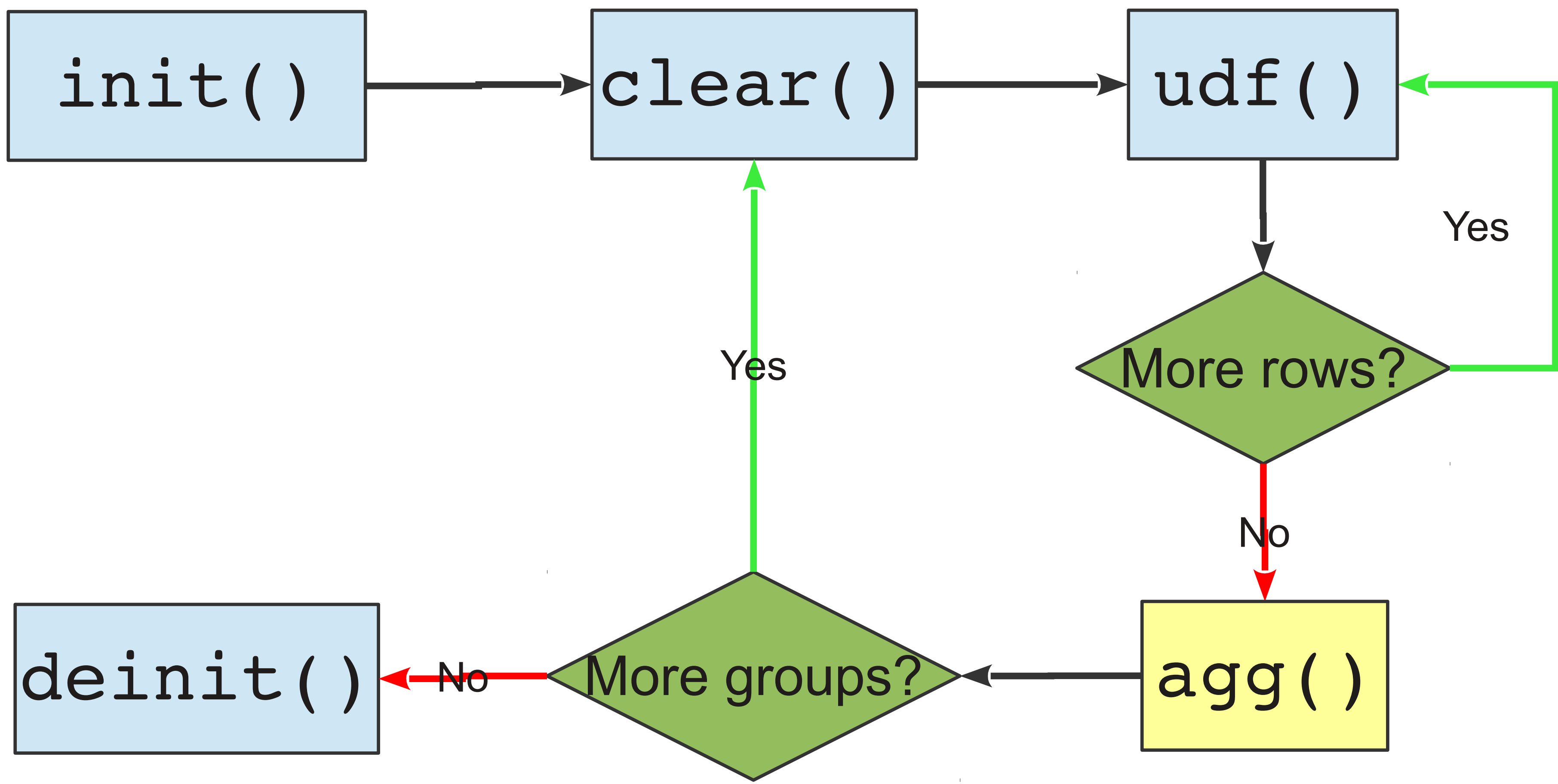
Data type Mapping

Type family	MySQL column data type	MYSQL UDF data type	v8 type	JS Type
Integral numbers	BIGINT INT MEDIUMINT SMALLINT TINYINT	INT_RESULT	v8::Integer or v8::Number	Number
Floating point numbers	DOUBLE FLOAT	REAL_RESULT	v8::Number	
Decimal numbers	DECIMAL	DECIMAL_RESULT		
Binary String	BINARY BLOB LONGBLOB MEDIUMBLOB VARBINARY TINYBLOB	STRING_RESULT	v8::String	String
Character String	CHAR LONGTEXT MEDIUMTEXT VARCHAR TEXT TINYTEXT			
Structured String	ENUM SET			
Temporal	DATE DATETIME TIME TIMESTAMP			

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



https://github.com/rpbouman/mysqlv8udfs

`jsagg()` example: JSON export

```
mysql> SELECT jsagg('
    ->   var rows, args = arguments, n = args.length;
    ->   function clear(){
    ->     console.info("clear");
    ->     rows = [];
    ->   }
    ->   function udf(){
    ->     console.info("udf");
    ->     var i, arg, row = {};
    ->     for (i = 0; i < n; i++){
    ->       arg = args[i];
    ->       row[arg.name] = arg.value;
    ->     }
    ->     rows.push(row);
    ->   }
    ->   function agg(){
    ->     console.info("agg");
    ->     return JSON.stringify(rows, null, "  ");
    ->   }
    -> ', film_id, title, release_year, description) AS json
    -> FROM sakila.film GROUP BY rating;
```

`jsagg()` example: result

```
[
 {
  "film_id": 1,
  "title": "ACADEMY DINOSAUR",
  "release_year": 2006,
  "description": "A Epic Drama of ... in The Canadian Rockies"
 },
 ...,
 ...,
 {
  "film_id": 1000,
  "title": "ZORRO ARK",
  "release_year": 2006,
  "description": "A Intrepid Panorama of ... in A Monastery"
 }
]
```

https://github.com/rpbouman/mysqlv8udfs

`jsagg()` example: error log

```
2013-09-16 23:36:45 JS_DAEMON [info]: Clear
2013-09-16 23:36:45 JS_DAEMON [info]: Udf
.... .. .. .. .. .. .. ...... .... ...
2013-09-16 23:36:45 JS_DAEMON [info]: Udf
2013-09-16 23:36:45 JS_DAEMON [info]: Agg
2013-09-16 23:36:45 JS_DAEMON [info]: Clear
2013-09-16 23:36:45 JS_DAEMON [info]: Udf
.... .. .. .. .. .. .. ...... .... ...
2013-09-16 23:36:45 JS_DAEMON [info]: Udf
2013-09-16 23:36:45 JS_DAEMON [info]: Agg
.... .. .. .. .. .. .. ...... .... ...
.... .. .. .. .. .. .. ...... .... ...
2013-09-16 23:36:45 JS_DAEMON [info]: Clear
2013-09-16 23:36:45 JS_DAEMON [info]: Udf
.... .. .. .. .. .. .. ...... .... ...
2013-09-16 23:36:45 JS_DAEMON [info]: Udf
2013-09-16 23:36:45 JS_DAEMON [info]: Agg
```

https://github.com/rpbouman/mysqlv8udfs

JavaScript Environment

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

`require()` example:

```
mysql> SELECT jsagg('
    ->   require("json_export.js")
    -> ', category_id, name) AS json
    -> FROM   sakila.category
```

```
[
 {
  "category_id": 1,
  "name": "Action"
 },
 ...,
 {
  "category_id": 16,
  "name": "Travel"
 }
]
```

https://github.com/rpbouman/mysqlv8udfs

require() example: json_export.js script

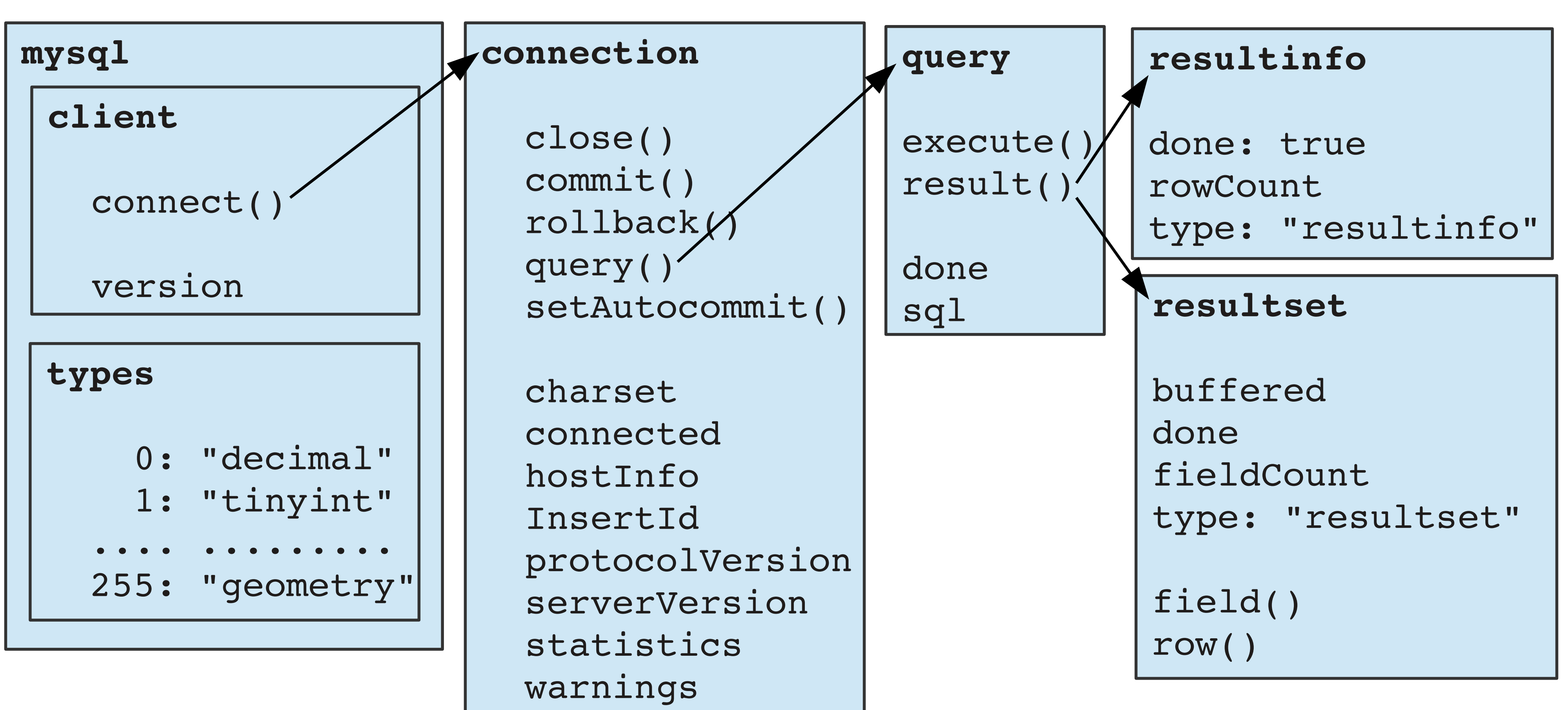
```
(function json_export(){
  var rows, row, i, arg, args = this.arguments, n = args.length;

  this.clear = function(){
    rows = [];
  }

  this.udf = function() {
    rows.push(row = {});
    for (i = 0; i < n; i++) {
      arg = args[i];
      row[arg.name] = arg.value;
    }
  }

  this.agg = function(){
    return JSON.stringify(rows, null, " ");
  }
})();
```

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



https://github.com/rpbouman/mysqlv8udfs

Mysql client example: inventory_held_by_customer

```
CREATE FUNCTION inventory_held_by_customer(p_inventory_id INT)
RETURNS INT
READS SQL DATA
BEGIN
  DECLARE v_customer_id INT;
  DECLARE EXIT HANDLER FOR NOT FOUND RETURN NULL;

  SELECT customer_id INTO v_customer_id
  FROM rental
  WHERE return_date IS NULL
  AND inventory_id = p_inventory_id;

  RETURN v_customer_id;
END;
```

https://github.com/rpbouman/mysqlv8udfs

Mysql client example

```
(function(){
  var conn;
  this.init = function(){
    var args = this.arguments;
    if (args.length !== 1 || args[0].type !== INT_RESULT) {
      throw "Single integer argument required";
    }
    conn = mysql.client.connect({
      user: "sakila",
      password: "sakila",
      schema: "sakila"
    });
  }
  this.udf = function(inventory_id){
    var query = conn.query(
      "SELECT customer_id FROM rental WHERE return_date IS NULL " +
      "AND inventory_id = " + inventory_id
    );
    query.execute();
    var result = query.result();
    if (result.done) return null;
    return result.row()[0];
  }
  this.deinit = function(){
    conn.close();
  }
})();
```

Finally...

- Fork it on github. I appreciate your interest!
  - https://github.com/rpbouman/mysqlv8udfs
  - https://github.com/rpbouman/mysqlv8udfs/wiki
- Sveta Smirnova's presentation on JSON UDF's
  - 2:30 – 3:30 PM, Taylor Suite
  - JSON UDFs available at http://labs.mysql.com/

Finally...

- Fork it on github. I appreciate your interest!
  - https://github.com/rpbouman/mysqlv8udfs
  - https://github.com/rpbouman/mysqlv8udfs/wiki
- Sveta Smirnova's JSON UDF's
  - available at http://labs.mysql.com/

Questions?

https://github.com/rpbouman/mysqlv8udfs