Perl Stored Procedures for MySQL

Using the Perl plugin for the External Language Stored Procedure framework.

Antony T Curtis <atcurtis@google.com>

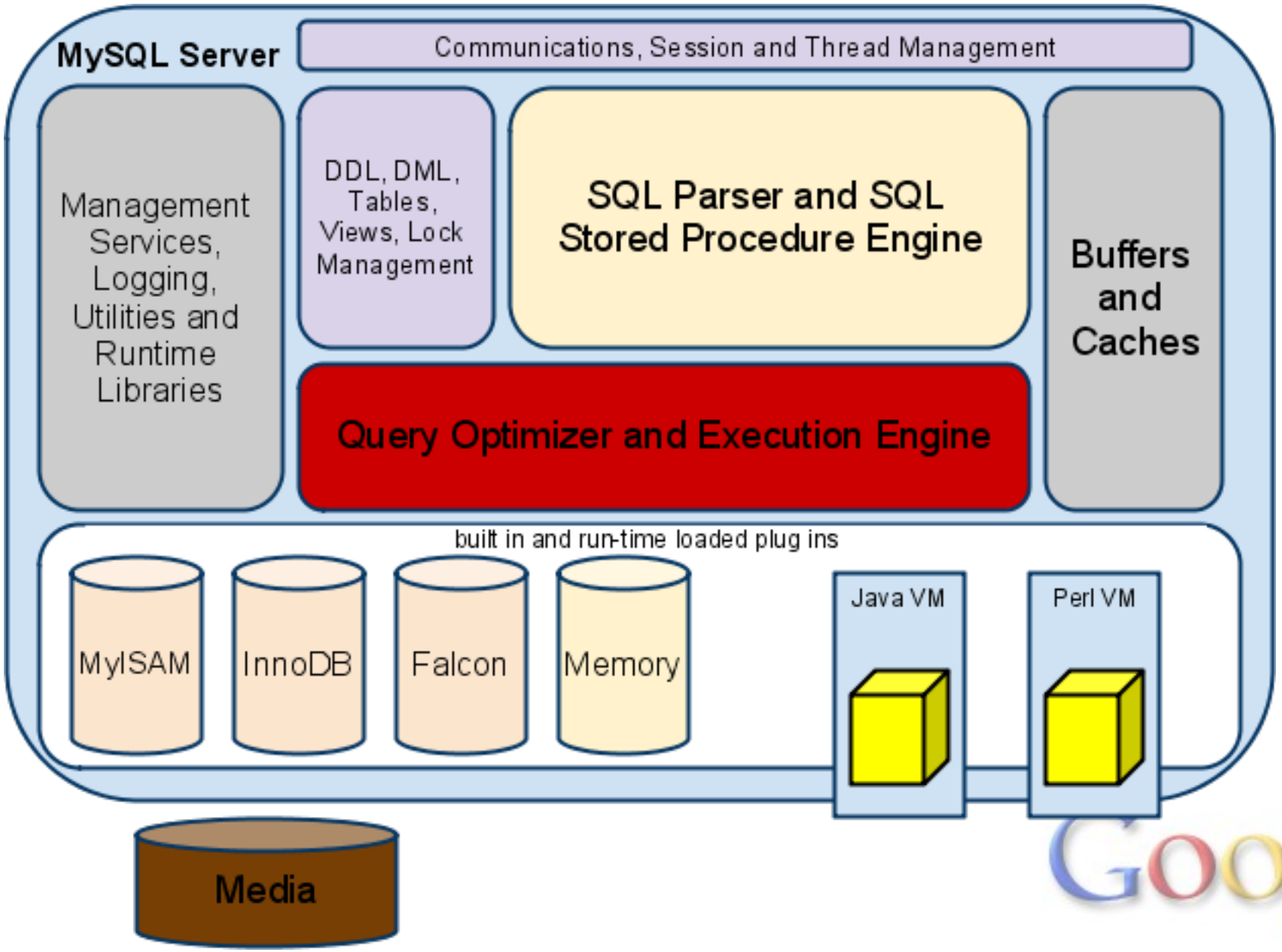
Google™

Why Perl?

- CPAN
- "Multiplicity" / Thread-friendly
- Lots of code already written
- Clean APIs - example: DBI
- MySQL UDFs have no access control
- MySQL UDFs cannot execute dynamic SQL
- Faster than MySQL's SQL Stored Procedures
- Complex code uses less memory than MySQL's SQL SP

Google™

Server Overview



Google™

Use the source,

Download the source tarball from
https://launchpad.net/mysql-wl820/trunk/5.1.33-wl820

```
# configure and build
BUILD/compile-pentium-debug-max-no-ndb --no-tune-cpu \
    --prefix=<install path>

# install
make install

# initialize db
cd <install path> ; bin/mysql_install_db
```

Google™

Preparing for the first time

Three ways to prepare the mysqld for Perl:

1. `INSTALL PLUGIN Perl SONAME "psm_perl.so";`
2. command line mysqld parameters
   ```
   $ libexec/mysqld \
       --plugin-load=perl=psm_perl.so
   ```
3. within the my.cnf config file

   ```
   [mysqld]
   plugin-load=perl=psm_perl.so
   ```

Google™

"Hello World"

```
mysql> CREATE PROCEDURE PerlHello()
    ->         DYNAMIC RESULT SETS 1
    ->         NO SQL
    ->         LANGUAGE Perl
    ->         EXTERNAL NAME "HelloWorld::test";
Query OK, 0 rows affected (0.00 sec)

mysql> CALL PerlHello;
+-----------------------+
| message               |
+-----------------------+
| Hello World from Perl |
+-----------------------+
1 row in set (0.02 sec)

Query OK, 0 rows affected (0.03 sec)
```

```
package HelloWorld;
# put this file in <prefix>/lib/mysql/perl
use 5.008008;
use strict;
use warnings;
use Symbol qw(delete_package);
require Exporter;
our @ISA = qw(Exporter);
our @EXPORT_OK = qw( );
our @EXPORT = qw( test );
our $VERSION = '0.01';


sub test()
{
  return {
    'message' => 'Hello World from Perl',
  };
}

1;
```

Features

- Uses Perl prototypes for simplicity
- threads don't block each other
- pass by value (`IN`) parameters
- pass-by-reference (`INOUT`/`OUT`) parameters
- simple auto-casting of values
- functions have simple single value result
- procedure may return single result set
  - array of hash
  - array of array
  - hash result (single row)
  - array result (single row)

Features, part 2

- detects if Perl module has changed and reloads
- dynamic SQL support using DBI and DBD::mysql
- `READS SQL DATA` attribute forces read-only mode
- `NO SQL` attribute blocks inline dynamic SQL.
- executes dynamic SQL in same thread/transaction
- `die` returns error message up to the client
- routines may be used in triggers and events
- procedures may be used as table functions

NOTE: Unmodified MySQL does not have table functions, yet!
No dynamic SQL support from triggers, table funcs or events.

Google™

Table Functions (example)

```
mysql> CREATE FUNCTION HelloFunc()
    ->         RETURNS TABLE (
    ->           message VARCHAR(64)
    ->         )
    ->         NO SQL
    ->         LANGUAGE Perl
    ->         EXTERNAL NAME "HelloWorld::test";
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM TABLE(HelloFunc) AS HelloFunc;
+-----------------------+
| message               |
+-----------------------+
| Hello World from Perl |
+-----------------------+
1 row in set (0.01 sec)
```

Note: Correlation name is mandatory for table functions.

Demonstration

Limitations and gotchas

- routines must not fork or create new threads
  - bad things happen - usually crashes.
- functions cannot access tables with dynamic SQL
  - limitation of MySQL's open_and_lock internals
- cannot detect changes in dependent modules
  - force reload by touching module time-stamp
- currently,no way to flush MySQL procedure/function cache
  - client connection must disconnect to 'forget'
- Runs "in-process" with mysqld
  - can cause mysqld to die from OOM error
  - can potentially corrupt mysqld internal data

Future Directions

- Inline declaration of Perl code
  - Store code in database
- "Fenced" mode
  - Perl modules would load in separate process
  - likely to use "process per user or security context"
- "Atomic" routines
  - create `SAVEPOINT` before entering routine
  - `ROLLBACK SAVEPOINT` on error.
- Fix/modify MySQL's table opening code (unlikely)
  - allow routines to open additional tables in read mode
  - would allow routines to query database without non-standard SQL syntax hacks

Google™

Resources...

- LaunchPad project
  https://launchpad.net/sakila-server/wl820

- Source tarballs
  https://launchpad.net/mysql-wl820/trunk/5.1.33-wl820

Perl sources used in demo will be uploaded.

- Modified PIE LaunchPad project
  https://code.launchpad.net/~atcurtis/pie/devel

Questions?

Antony T Curtis <atcurtis@google.com>

Google™

More Perl Examples

```
# CREATE FUNCTION error_check(
#   error_code INT,
#   error_message VARCHAR(512)
# ) RETURNS INT
# NO SQL
# LANGUAGE Perl
# EXTERNAL NAME "...::error_check"


sub error_check($$)
{
  my ($errcode, $errmsg) = @_;
  die "[$errcode] $errmsg"
    if $errcode;
  return $errcode;
}
```

```
# CREATE PROCEDURE check_data()
# READS SQL DATA
# LANGUAGE Perl
# EXTERNAL NAME "...::check_data"

sub check_data()
{
  my $dbh = DBI->connect(
    "dbi:mysql:test", undef, undef)
  or die "Failed to connect";

  # attempts to update/write data
  # using this connection will fail.

  ...

}
```

More Perl Examples, part 2

```
# CREATE FUNCTION my_reverse(
#   string TEXT
# ) RETURNS TEXT
# NO SQL
# LANGUAGE Perl
# EXTERNAL NAME
#   "MyUDFExample::my_reverse";

sub my_reverse($)
{
  my($string)= @_;
#  print "foo\n";
  return reverse $string;
}
```

```
# CREATE PROCEDURE my_reverseproc(
#   INOUT string TEXT
# )
# NO SQL
# LANGUAGE Perl
# EXTERNAL NAME
#   "MyUDFExample::my_reverseproc";

sub my_reverseproc(\$)
{
  my($stringref)= @_;
  $$stringref= reverse $$stringref;
  # no resultsets
  return undef;
}
```