External Language Stored Procedures Framework for MySQL

Antony T Curtis <atcurtis@google.com>

Project history...

- Started as a UDF-ng project inside MySQL in 2006.
- Primary collaborator - Eric Herman.
- First functional alpha in 2007 (Heidelberg).
- Public "unsupported" beta release in UC 2008.
- Maintained and enhanced using Google 20% time.
- Plug-ins developed for:
  - Java (non-fenced)
  - Perl (non-fenced)
  - XML-RPC

https://launchpad.net/sakila-server/wl820
https://launchpad.net/mysql-wl820/trunk/5.1.33-wl820

Google™

But we already have UDFs...

User Defined Function

- No access control.
- Only C/C++ supported.
- No dynamic SQL
- No support for result-sets
- Supports varadic params
- Simple value results
- Cannot use as table
- Aggregate functions

External Stored Procedure

- Fine grained ACLs
- C/C++, Java, Perl (so far)
- Supports dynamic SQL
- Supports multiple result-sets
- No varadic support
- Casted value results
- Supports table functions
- No aggregates (workarounds exist)

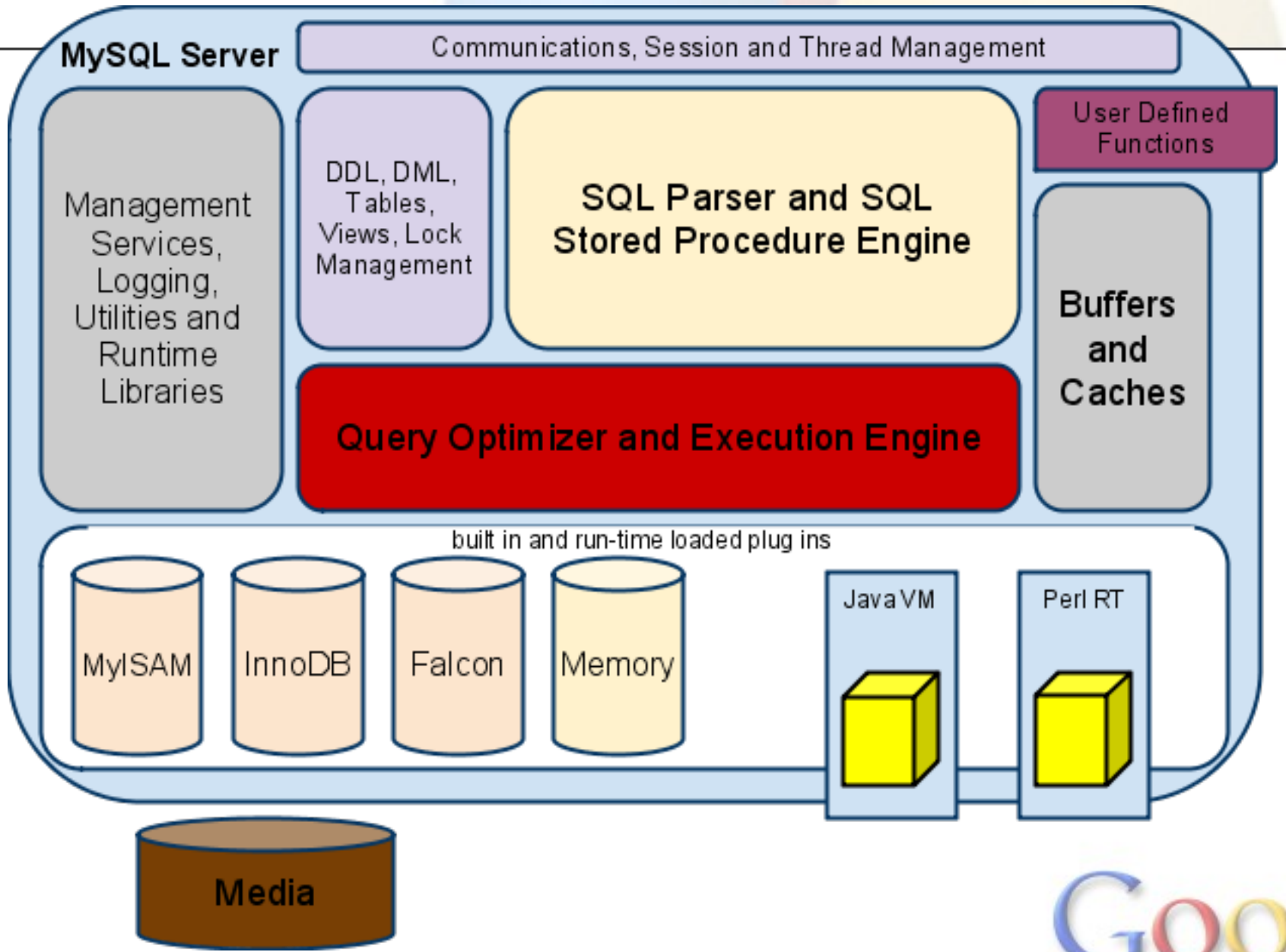
Google™

Implementation Objectives

- Few changes to the parser
  - Closer to SQL standards
- Minimal changes to the stored procedure engine
  - Only one new instruction class
- Minor change to system tables
  - Change 'language' column from ENUM to VARCHAR
- Refactor Protocol classes
  - Avoid using preprocessor, use object orientation
  - Use embedded 'protocol' for dynamic SQL
- Minor change to libmysql client libraries
  - add less than 10 lines of code
- **Keep server changes small / low impact**

Google™

Server Overview



Google™

Implementation

MySQL Stored Procedure execution "instruction"

```
class sp_instr ...
{
  /**
    Execute this instruction
    @param thd          Thread handle
    @param[out] nextp   index of the next instruction to execute. (For most
                        instructions this will be the instruction following this
                        one). Note that this parameter is undefined in case of
                        errors, use get_cont_dest() to find the continuation
                        instruction for CONTINUE error handlers.
    @retval 0       on success,
    @retval other   if some error occured
  */

  virtual int execute(THD *thd, select_result *result, uint *nextp) = 0;
};
```

New subclass for external stored procedures:

```
class sp_instr_external
```

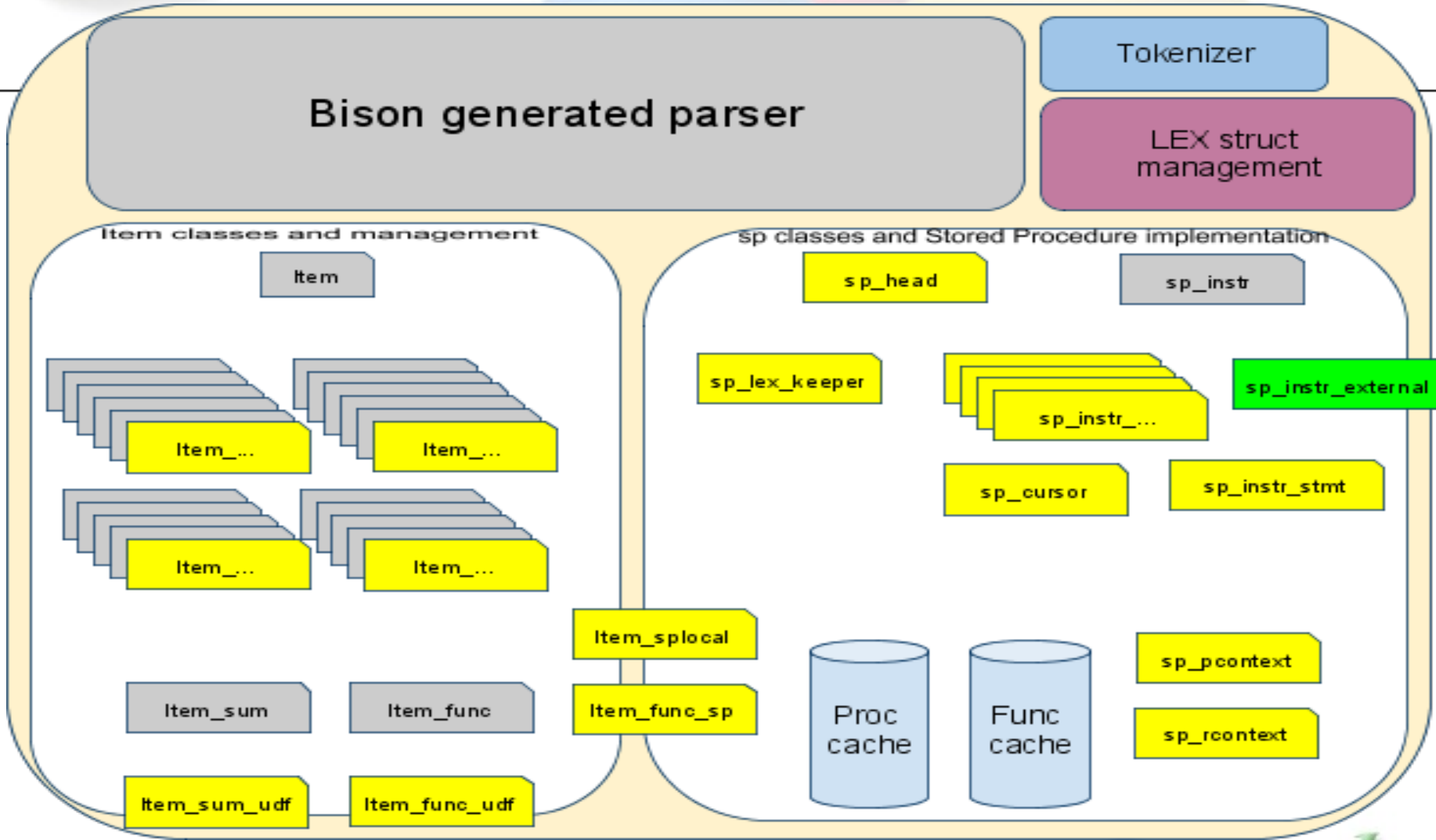
Table Functions

```
bool mysql_derived_filling(THD *thd, LEX *lex, TABLE_LIST *orig_table_list)
{
  ...

  /*check that table creation pass without problem and it is derived table */
  if (table && unit)
  {
    ...

    if (unit->is_union())
    {
      // execute union without clean up
      res= unit->exec();
    }
    else
    if (unit->is_table_function())
    {
      ...
      res= derived_result->prepare(unit->types, unit) ||
           unit->sp->execute_procedure(thd, &args, derived_result);
    }
    else
    {
      ...
```

Server Parser Overview



Google™

Plug-in Essentials

```
/* Plugin type-specific descriptor */

static struct st_mysql_psmlanguage example_psm_descriptor=
{
  MYSQL_PSM_LANGUAGE_INTERFACE_VERSION, /* interface version */
  example_psm_find,                     /* function resolution function */
  example_psm_release,                  /* function release function */
  example_psm_execute                   /* execute function */
};

/* Plugin library descriptor */

mysql_declare_plugin(example)
{
  MYSQL_PSMLANGUAGE_PLUGIN,             /* type */
  &example_psm_descriptor,              /* descriptor */
  "Example",                            /* name */

   ...
```

Plug-in Essentials, part 2

```
struct st_mysql_psmcontext
{
...
  inline int store_null(int idx);
  inline int store_string(int idx, const char *str, int length,
                          struct charset_info_st *cs);
  inline int store_double(int idx, double nr, int precision);
  inline int store_integer(int idx, long long nr, char unsigned_val);
  inline int store_time(int idx, struct st_mysql_time *ltime);
  inline int val_null(int idx);
  inline const char *val_string(int idx, char *str, int *length,
                                struct charset_info_st **cs);
  inline long long val_integer(int idx);
  inline double val_double(int idx);

  inline int field_ptr(int idx, int *field_type, void **ptr, int *length);

  inline int row_field(const char *title, int field_type,
                       int size, int precision);
  inline int row_prepare();
  inline int row_send();
  inline int row_send_eof();
...
}
```

Declaring routines

New/changed routine attributes:

- LANGUAGE <identifier>
- EXTERNAL NAME <string-literal>
- DYNAMIC RESULT SETS <numeric-literal>

Function return type extension:

- TABLE (<field-declaration> ...)

Google™

Dynamic SQL support

Minimal change to libmysqlclient library - No change to ABI.

```
MYSQL * STDCALL
CLI_MYSQL_REAL_CONNECT(MYSQL *mysql,const char *host, const char *user,
                       const char *passwd, const char *db,
                       uint port, const char *unix_socket,ulong client_flag)
{
  ...
  DBUG_ENTER("mysql_real_connect");

#if defined(HAVE_CC_WEAK_ATTRIBUTE) || defined(HAVE_CC_WEAK_PRAGMA)
  if (ext_mysql_real_connect &&
      mysql->options.methods_to_use != MYSQL_OPT_USE_REMOTE_CONNECTION)
  {
    ...
    if (mysql->options.methods_to_use == MYSQL_OPT_USE_INLINE_CONNECTION ||
        (mysql->options.methods_to_use == MYSQL_OPT_GUESS_CONNECTION &&
         (!host || !*host || !strcmp(host,LOCAL_HOST))))
      DBUG_RETURN(ext_mysql_real_connect(mysql, host, user,
                                         passwd, db, port,
                                         unix_socket, client_flag));
  }
#endif

  /* Don't give sigpipe errors if the client doesn't want them */
```

Dynamic SQL support, part 2

- External routines can use dynamic SQL simply by using libmysqlclient library.

Example:
Using Perl, you can use dynamic SQL simply by using DBD:: mysql as is.

```
sub test1()
{
  my $dsn= "DBI:mysql:test";
  my $dbh= DBI->connect($dsn, undef, undef) or die "Failed $!";

  $dbh->do("INSERT INTO test.t1 (txt) VALUES ('hello world')");

  # This routine has no resultsets
  return undef;
}
```

Google™

Example: XML-RPC Routines

```
mysql> create function xml_get_state(id int)
    -> returns text
    -> no sql
    -> language xmlrpc
    -> external name
    -> 'xmlrpc://betty.userland.com/RPC2;examples.getStateName';
Query OK, 0 rows affected (0.00 sec)

mysql> select xml_get_state(40);
+--------------------+
| xml_get_state(40)  |
+--------------------+
| South Carolina     |
+--------------------+
1 row in set (0.42 sec)
```

Google™

Example: XML-RPC Routines, part 2

- find_routine: parse and validate supplied URL and return a handle for the xmlrpc service
- exec_routine: invoke xmlrpc service and then marshall the result back to mysql.

Code is in `/plugins/psm_xmlrpc/`

Brief code walk-through...

Google™

Future directions

- Support dynamic SQL from Java
  (work underway by Eric Herman)

- Support more languages - Python, Lua, PHP?
- Support dynamically declared routines... example:

```
CREATE FUNCTION get_env(arg varchar(32))
RETURNS TABLE (
  `key` VARCHAR(128),
  `value` VARCHAR(2048)
)
LANGUAGE Perl
AS '
  my @result = ();
  foreach my $var (keys %ENV) {
    next if defined($arg) && !($key =~ m/$arg/);
    my %row = ( key=>$var, value=>$ENV{$var} );
  push @result, \%row;
}
return \@result;';
```

Google™

Questions?

Antony T Curtis <atcurtis@google.com>