Creating a new function in MariaDB

Michael Widenius
CTO @ MariaDB

Compile MariaDB for easy development

• Get the source code (tar file, git, linux distribution etc)
• Compile
>./BUILD/compile-pentium64-debug-max

• Add to your ~/.my.cnf:
[mariadb-10.5]
datadir=/my/data105
lc-messages-dir=/my/maria-10.5/sql/share/english

• Create the default databases
>./scripts/mysql_install_db

• Start mysqld in your debugger:
> cd sql ; ddd mysqld ; run

MariaDB

Files that stores expression code

```
> cd sql
> ls item*
item.cc           item_inetfunc.cc   item_sum.h
item.h            item_inetfunc.h    item_timefunc.cc
item_buff.cc      item_jsonfunc.cc   item_timefunc.h
item_cmpfunc.cc   item_jsonfunc.h    item_vers.cc
item_cmpfunc.h    item_row.cc        item_vers.h
item_create.cc    item_row.h         item_windowfunc.cc
item_create.h     item_strfunc.cc    item_windowfunc.h
item_func.cc      item_strfunc.h     item_xmlfunc.cc
item_func.h       item_subselect.cc  item_xmlfunc.h
item_geofunc.cc   item_subselect.h
item_geofunc.h    item_sum.cc
```

Files to change when adding a new SQL function

In most cases one has only to change some of the following files:

sql_yacc.yy and sql_yacc_ora.yy

- Not needed for simple functions that takes a fix number of arguments
- Needed for operators and group functions (as group functions may need the DISTINCT keyword)
- If the function name is a SQL keyword (LEFT, RIGHT etc)

item_create.cc

- Add some interface code to create the item that will execute the function

Item_*.cc and item_*.h

MariaDB

Finding similar code

Start by looking at some existing function similar to what you want to do:

```
> grep -n func_left item* *yacc*yy
item_strfunc.cc:1630:String *Item_func_left::val_str(String *str)
item_strfunc.cc:1666:bool Item_func_left::fix_length_and_dec()
item_strfunc.h:454:class Item_func_left :public Item_str_func
item_strfunc.h:458:  Item_func_left(THD *thd, Item *a, Item *b):
Item_str_func(thd, a, b) {}
item_strfunc.h:463:  { return get_item_copy<Item_func_left>(thd,
this); }
sql_yacc.yy:10541:            $$= new (thd->mem_root)
Item_func_left(thd, $3, $5);
sql_yacc_ora.yy:10641:            $$= new (thd->mem_root)
```

PI()
item_create.cc part 1

```
class Create_func_pi : public Create_func_arg0
{
public:
  virtual Item *create_builder(THD *thd);
  static Create_func_pi s_singleton;
protected:
  Create_func_pi() {}
  virtual ~Create_func_pi() {}
};

Create_func_pi Create_func_pi::s_singleton;
```

PI()
item_create.cc part 2

```
Item*
Create_func_pi::create_builder(THD *thd)
{
  return new (thd->mem_root) Item_static_float_func(thd, "pi()",
M_PI, 6, 8);
}
....

static Native_func_registry func_array[] =
{
  { { STRING_WITH_LEN("PI") }, BUILDER(Create_func_pi)},
```

ABS(X)
item_create.cc part 1

```
class Create_func_abs : public Create_func_arg1
{
public:
  virtual Item *create_1_arg(THD *thd, Item *arg1);
  static Create_func_abs s_singleton;

protected:
  Create_func_abs() {}
  virtual ~Create_func_abs() {}
};

Create_func_abs Create_func_abs::s_singleton;
```

MariaDB

ABS(X)
item_create.cc part 2

```
Item*
Create_func_abs::create_1_arg(THD *thd, Item *arg1)
{
  return new (thd->mem_root) Item_func_abs(thd, arg1);
}


...
static Native_func_registry func_array[] =
{
  { { STRING_WITH_LEN("ABS") },
BUILDER(Create_func_abs)},
```

ABS(X)
item_func.h

```
class Item_func_abs :public Item_func_num1
{
public:
  Item_func_abs(THD *thd, Item *a): Item_func_num1(thd, a) {}
  double real_op();
  longlong int_op();
  my_decimal *decimal_op(my_decimal *);
  const char *func_name() const { return "abs"; }
  void fix_length_and_dec_int();
  void fix_length_and_dec_double();
  void fix_length_and_dec_decimal();
  bool fix_length_and_dec();
  Item *get_copy(THD *thd)
  { return get_item_copy<Item_func_abs>(thd, this); }
};
```

MariaDB

ABS(X)
item_func.cc::fix_length...int

```
void Item_func_abs::fix_length_and_dec_int()
{
  max_length= args[0]->max_length;
  unsigned_flag= args[0]->unsigned_flag;
  /*
    type_handler_long_or_longlong set's a different type handler
depending
    on the argument is long or longlong and if signed or not
 */
  set_handler(type_handler_long_or_longlong());
}
```

ABS(X)
item_func.cc::fix_length...double

```
void Item_func_abs::fix_length_and_dec_double()
{
  set_handler(&type_handler_double);
  decimals= args[0]->decimals; // Preserve NOT_FIXED_DEC
  max_length= float_length(decimals);
  unsigned_flag= args[0]->unsigned_flag;
}
```

ABS(X)
item_func.cc::real_op()

```
double Item_func_abs::real_op()
{
  double value= args[0]->val_real();
  null_value= args[0]->null_value;
  return fabs(value);
}
```

MariaDB

ABS(X)
item_func.cc::int_op()

```
longlong Item_func_abs::int_op()
{
  longlong value= args[0]->val_int();
  if ((null_value= args[0]->null_value))
    return 0;
  if (unsigned_flag)
    return value;
  /* -LONGLONG_MIN = LONGLONG_MAX + 1 => outside of signed
longlong range */
  if (value == LONGLONG_MIN)
    return raise_integer_overflow();
  return (value >= 0) ? value : -value;
}
```

MariaDB

LEFT(str,length)
sql_yacc.yy

LEFT is a sql keyword, which is why it’s create is defined in sql_yacc.yy

```
| LEFT '(' expr ',' expr ')'
  {
    $$= new (thd->mem_root) Item_func_left(thd, $3, $5);
    if (unlikely($$ == NULL))
      MYSQL_YYABORT;
  }
```

LEFT(str,length)
item_strfunc.h

```
class Item_func_left :public Item_str_func
{
  String tmp_value;
public:
  Item_func_left(THD *thd, Item *a, Item *b): Item_str_func(thd, a, b)
{}
  String *val_str(String *);
  bool fix_length_and_dec();
  const char *func_name() const { return "left"; }
  Item *get_copy(THD *thd)
  { return get_item_copy<Item_func_left>(thd, this); }
};
```

MariaDB

LEFT(str,length)
item_strfunc.cc::fix_length

```
bool Item_func_left::fix_length_and_dec()
{
  if (agg_arg_charsets_for_string_result(collation, args, 1))
    return TRUE;
  DBUG_ASSERT(collation.collation != NULL);
  left_right_max_length();
  return FALSE;
}
```

LEFT(str,length)
item_strfunc.cc::fix_length

```
void Item_str_func::left_right_max_length()
{
  uint32 char_length= args[0]->max_char_length();
  if (args[1]->const_item() && !args[1]->is_expensive())
  {
    Repeat_count tmp(args[1]);
    set_if_smaller(char_length, (uint) tmp.count());
  }
  fix_char_length(char_length);
}
```

MariaDB

LEFT(str,length)
item_strfunc.cc::val_str() part 1

```
String *Item_func_left::val_str(String *str)
{
  DBUG_ASSERT(fixed == 1);
  String *res= args[0]->val_str(str);

  /* must be longlong to avoid truncation */
  longlong length= args[1]->val_int();
  uint char_pos;

  if ((null_value=(args[0]->null_value || args[1]->null_value)))
    return 0;
```

LEFT(str,length)
item_strfunc.cc::val_str()

```
  /* if "unsigned_flag" is set, we have a *huge* positive number. */
  if ((length <= 0) && (!args[1]->unsigned_flag))
    return make_empty_result();
  if ((res->length() <= (ulonglong) length) ||
      (res->length() <= (char_pos= res->charpos((int) length))))
    return res;

  tmp_value.set(*res, 0, char_pos);
  return &tmp_value;
}
```

MariaDB

ELT(pos,arg1,...)
item_create.cc part 1

```
class Create_func_elt : public Create_native_func
{
public:
  virtual Item *create_native(THD *thd, LEX_CSTRING *name,
List<Item> *item_list);
  static Create_func_elt s_singleton;

protected:
  Create_func_elt() {}
  virtual ~Create_func_elt() {}
};

Create_func_elt Create_func_elt::s_singleton;
```

MariaDB

ELT(pos,arg1,...)
item_create.cc part 2

```
Item* Create_func_elt::create_native(THD *thd, LEX_CSTRING
*name,
                                     List<Item> *item_list)
{
  int arg_count= 0;
  if (item_list != NULL)
    arg_count= item_list->elements;
  if (unlikely(arg_count < 2))
  {
    my_error(ER_WRONG_PARAMCOUNT_TO_NATIVE_FCT, MYF(0),
name->str);
    return NULL;
  }
  return new (thd->mem_root) Item_func_elt(thd, *item_list);
```

MariaDB

ELT(pos,arg1,...)
item_strfunc.h

```
class Item_func_elt :public Item_str_func
{
public:
  Item_func_elt(THD *thd, List<Item> &list): Item_str_func(thd,
list) {}
  double val_real();
  longlong val_int();
  String *val_str(String *str);
  bool fix_length_and_dec();
  const char *func_name() const { return "elt"; }
  Item *get_copy(THD *thd)
  { return get_item_copy<Item_func_elt>(thd, this); }
};
```

MariaDB

ELT(pos,arg1,...)
item_strfunc.cc: fix_length

```
bool Item_func_elt::fix_length_and_dec()
{
  uint32 char_length= 0;
  decimals=0;
  if (agg_arg_charsets_for_string_result(collation, args + 1,
arg_count - 1))
    return TRUE;
  for (uint i= 1 ; i < arg_count ; i++)
  {
    set_if_bigger(char_length, args[i]->max_char_length());
    set_if_bigger(decimals,args[i]->decimals);
  }
  fix_char_length(char_length);
  maybe_null=1;                 // NULL if wrong first arg
```

ELT(pos,arg1,...)
item_strfunc.cc::val_str()

```
String *Item_func_elt::val_str(String *str)
{
  DBUG_ASSERT(fixed == 1);
  uint tmp;
  null_value=1;
  if ((tmp=(uint) args[0]->val_int()) == 0 || tmp >= arg_count)
    return NULL;

  String *result= args[tmp]->val_str(str);
  if (result)
    result->set_charset(collation.collation);
  null_value= args[tmp]->null_value;
  return result;
}
```

MariaDB

ELT(pos,arg1,...)
item_strfunc.cc::val_real()

```
double Item_func_elt::val_real()
{
  DBUG_ASSERT(fixed == 1);
  uint tmp;
  null_value=1;
  if ((tmp=(uint) args[0]->val_int()) == 0 || tmp >= arg_count)
    return 0.0;
  double result= args[tmp]->val_real();
  null_value= args[tmp]->null_value;
  return result;
}
```

Thank you