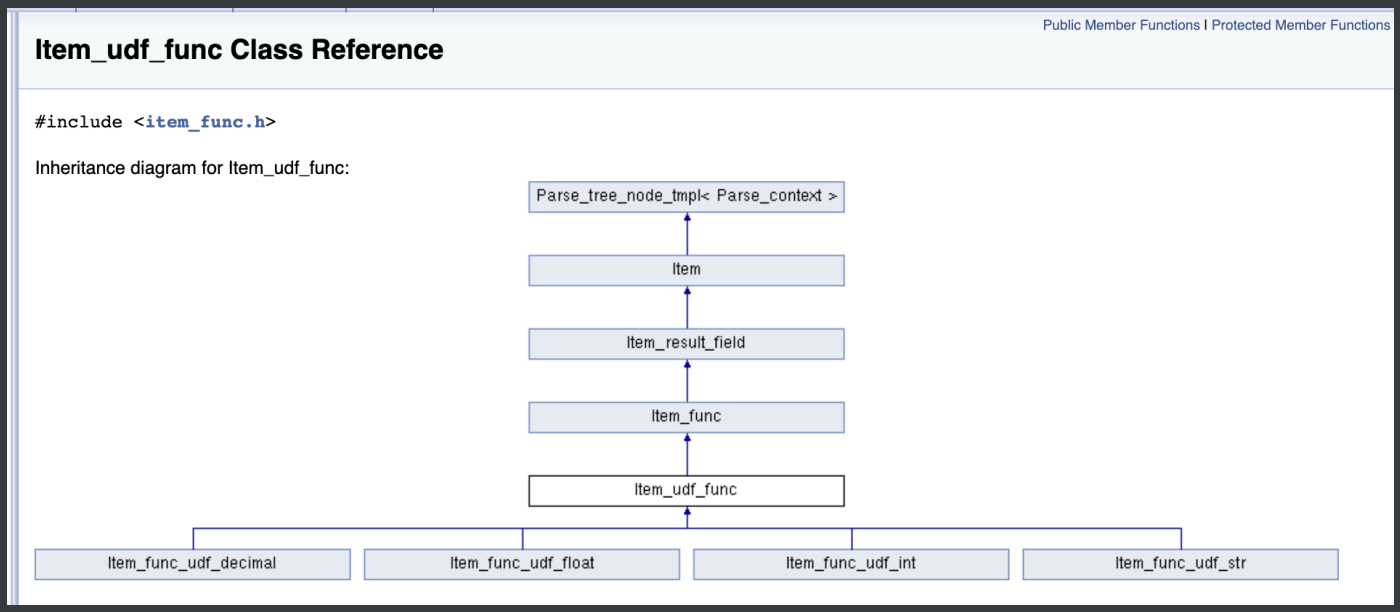
CryptDB UDF生成方法

UDF在SQL语句中的解析结构

参见MySQL官方文档，其为 `Item_udf_func` 类型。



根据返回值的类型，又可以额外分成decimal、float、int和str类型。

想要添加UDF，首先在你想用到的 `.cc` 文件中定义一个静态UDF struct结构

```
static udf_func udf_name {
    LEXSTRING("udf_name"),    // 你的UDF函数名字 这个宏定义如果没有的话，可以手动加到头文件中去。*
    STRING_RESULT,             // 返回值类型 可以是STRING_RESULT, INT_RESULT, REAL_RESULT...
    UDFTYPE_FUNCTION,          // 必须是UDFTYPE_FUNCTION
    NULL,                       // NULL就行
    NULL,                       // 同上
    NULL,                       // 同上
    NULL,                       // 同上
    NULL,                       // 同上
    NULL,                       // 同上
    NULL,                       // 同上
    0L,                         // 序列化id，默认0就行
};
```

*LEXSTRING定义：

```
#define LEXSTRING(cstr)           \
    {                             \
        (char *)cstr, sizeof(cstr) \
    }
```

随后在SQL语法树中创建一个 `Item_udf_func` 类型

示例：

```
List<Item> l;                          // list是必须的，它用来存参数，顺序和push的顺序一致（有点忘了，可测试一下）
                                        // 只接受泛型的指针作为内容
l.push_back(new Item_int(2));         // push进入一个参数2
l.push_back(copyWithTHD(&(param)))    // 利用内置copy函数push一个参数
Item_func_udf *const udf_function =
    new (current_thd->mem_root) Item_func_udf_str(&u_sum_a, l);
```

注意：

- 最好都用指针，因为MySQL parser中大部分的类都是抽象类，所以你不能直接用Item_func_udf来存。
- 子类可以转父类，如：

```
Item_udf_func_str *const child = ...;
Item *const parent = static_cast<Item *const>(child);
```

- `copyWithTHD` 函数可以原地复制一个 `const` 的指针 / 类出来，比较好用。

一个完整的改写例子

例如，有一句SQL指令

```
SELECT * FROM test WHERE enc_id = 123 OR nenc_id = 123;
```

会被解析成一个若干个MySQL语法树结构，随后根据lex特征，分发给派遣函数。其中有

`SELECT_LEX` 标志其为 `SELECT` 类型的语句，会被 `dml_handler` 中 `SelectHandler` 识别，并进行 `gather` 和 `rewrite` 操作。

最后这个语句中WHERE条件会被解析成MySQL语法树中的 `Item_cond_or` 结构表示其为OR类型的conditional语句，而 `Item_cond_or` 有两个参数，都是 `Item_func` 中的 `Item_func_eq` 结构。那么重写会在 `rewriter_func.cc` 文件中实现对func类型和cond类型的改写：

```
template <Item_func::Functype FT, class IT>
class CItemCond : public CItemSubtypeFT<Item_cond, FT> // 注意看CItemCond类
{
    virtual RewritePlan *
    do_gather_type(const Item_cond &i, Analysis &a) const // 对cond进行gather
    {
```

```
7           // 获取cond中的分量。我们的例子中为enc_id = 123 和 nenc_id = 123
8           const unsigned int arg_count =
9               RiboldMYSQL::argument_list(i)->elements;
10
11          const EncSet out_es = PLAIN_EncSet;
12          EncSet child_es = EQ_EncSet;
13
14          std::vector<std::pair<std::shared_ptr<RewritePlan>, OLK>>
15              out_child_olks(arg_count);
16
17          auto it =
18              RiboldMYSQL::constList_iterator<Item>
    (*RiboldMYSQL::argument_list(i));
19          unsigned int index = 0;
20          for (;;)
21          {
22              const Item *const argitem = it++;
23              if (!argitem)
24                  break;
25              assert(index < arg_count);
26
27              std::shared_ptr<RewritePlan>
28                  temp_childrp(gather(*argitem, a));
29              const OLK &olk =
30                  EQ_EncSet.intersect(temp_childrp->es_out).chooseOne();
31              out_child_olks[index] = std::make_pair(temp_childrp, olk);
32              ++index;
33          }
34
35          const std::string why = "and/or";
36          const reason rsn(out_es, why, i);
37
38          // Must be an OLK for each argument.
39          return new RewritePlanPerChildOLK(out_es, out_child_olks, rsn);
40      }
41
```

```
42      virtual Item *
43      do_rewrite_type(const Item_cond &i, const OLK &olk,
44                      const RewritePlan &rp, Analysis &a) const       //
    对cond进行重写，主要是对其分量的改写
45      {
46          const unsigned int arg_count =
47              RiboldMYSQL::argument_list(i)->elements;
48
49          const RewritePlanPerChildOLK &rp_per_child =
50              static_cast<const RewritePlanPerChildOLK &>(rp);
51          auto it = RiboldMYSQL::constList_iterator<Item>
    (*RiboldMYSQL::argument_list(i));
52          List<Item> out_list;
53          unsigned int index = 0;
54          for (;;)
55          {
56              const Item *const argitem = it++;
57              if (!argitem)
58              {
59                  break;
60              }
61              assert(index < arg_count);
62
63              const std::pair<std::shared_ptr<RewritePlan>, OLK>
64                  &rp_olk = rp_per_child.child_olks[index];
65              const std::shared_ptr<RewritePlan> &c_rp =
66                  rp_olk.first;
67              const OLK &olk = rp_olk.second;
68              Item *const out_item =
69                  itemTypes.do_rewrite(*argitem, olk, *c_rp.get(), a);
    // 注意这里，就是改写分量!
70              out_item->name = NULL;
71              out_list.push_back(out_item);
72              ++index;
73          }
74
```

```
        return new IT(out_list);
    }
};
```

而比较类型的（= != <= >= > <）func会在这里改写：

```
template <Item_func::Functype FT, class IT>
class CItemCompare : public CItemSubtypeFT<Item_func, FT>
{
    virtual Item *
    do_rewrite_type(const Item_func &i, const OLK &constr,    // 此处我实现了基于FH-OPE的udf改写
                    const RewritePlan &rp, Analysis &a)
        const
    {
        LOG(cdb_v) << "do_rewrite_type Item_func " << i << " constr "
                   << EncSet(constr) << std::endl;
        std::cout << "do_rewrite_type Item_func " << i << std::endl;
        TEST_BadItemArgumentCount(i.type(), 2, i.argument_count());

        /**
         * If this is an inequality comparison function and its arguments contain frequency-hiding columns,
         * we should rewrite the right arguments to a udf type.
         */

        Item *const *const args = i.arguments();
        const std::string &field_name = args[0]->name;

        /*
            特别注意这里，我通过列名判断了是否需要ope改写。
        */
        if (isInequalityFunc(i) && needFrequencySmoothing(field_name))
        {
            //std::cout << "I am here!\n";
```

```
28              return rewrite_args_fh_ope(static_cast<const Item_func *>
    (&i), a); // 此函数返回一个udf类型。
29          }
30
31          return rewrite_args_FN(i, constr,
32                                 static_cast<const RewritePlanOneOLK &>
    (rp),
33                                 a);
34      }
35  }
```

rewrite_args_fh_ope的实现函数为：

```
1   /*
2       返回一个udf指针。
3   */
4   Item *
5   getFHSerachUDF(const unsigned int &val)
6   {
7       List<Item> l = List<Item>();
8       Item *const arg = new Item_int((unsigned long long)val);
9       l.push_back(arg);
10
11      return new Item_func_udf_int(&u_fhsearch, l); // 返回int类型的udf
12  }
13
14  /*
15      说明：”FH-OPE改写是这样的： column < 3 会改成 column <= FHSearch(3,
    pos);也就是说，Item_func还是不变，但是其中一个       分量3会被改成UDF，因此最后
    我还是包装成了一个func类型。如果你不需要，就不用这个函数。
16  */
17  Item *
18  getNewFunc(Item *const lhs, Item *const rhs, const Item_func::Functype
    &type, bool found)
19  {
```

```
20      if (found)
21      {
22          switch (type)
23          {
24          case Item_func::Functype::LT_FUNC:
25              return new Item_func_lt(lhs, rhs);
26          case Item_func::Functype::LE_FUNC:
27              return new Item_func_le(lhs, rhs);
28          case Item_func::Functype::GT_FUNC:
29              return new Item_func_gt(lhs, rhs);
30          case Item_func::Functype::GE_FUNC:
31              return new Item_func_ge(lhs, rhs);
32
33          default:
34              UNIMPLEMENTED;
35          }
36      }
37      else
38      {
39          switch (type)
40          {
41          case Item_func::Functype::LT_FUNC:
42          case Item_func::Functype::LE_FUNC:
43              return new Item_func_le(lhs, rhs);
44          case Item_func::Functype::GT_FUNC:
45          case Item_func::Functype::GE_FUNC:
46              return new Item_func_gt(lhs, rhs);
47
48          default:
49              UNIMPLEMENTED;
50          }
51      }
52  }
53
54  Item *
55  rewrite_args_fh_ope(const Item_func *const item, Analysis &a)
```

```
{
    auto ret = find_pos();
    unsigned int pos = ret.first;
    bool found = ret.second;

    Item *const udf = getFHSerachUDF(pos);

    /**
     * Rewrite the second argument.
     *
     * We should get the anon_onion_name for FHOPE columns.
     * 根据具体实现以下代码可以不需要。
     */

    const FieldMeta &fm = a.getFieldMeta(db_name, table_name,
field_name);
    OnionMeta *const om = fm.getOnionMeta(onion::oFHOPE);
    const std::string anon_field_name_encoding = om->getAnonOnionName()
+ "_encoding";
    const std::string anon_table_name = a.getAnonTableName(db_name,
table_name);

    Item_field *const anon_item_field =
        make_item_field(*item_field, anon_table_name,
anon_field_name_encoding);

    return getNewFunc(anon_item_field, udf, item->functype(), found);
}
```