

关于where谓词的重写方法&update改法 + 解密不乱码

函数调用顺序：

- main;
- handle_line;
- executeQuery;
- queryPreamble;
- Rewriter::rewrite;
- dispatchOnLex;
- DML_Handler::transformLex;
- **XXXHandler::gather;**
- process_select_lex;
- **process_filters_lex;**
- gatherAndAddAnalysisRewritePlan;
- gather
- CItemTypedie<Item Type>do_gather
- CItemCompare::do_gather_type:

一个粗略的想法，不知道是不是都是可以的，反正判断大小都是可以的，但是 `in`，`not in` 和别的还没试过，至于 `having` 的思路是一样的。

```
template<Item_func::FuncType FT, class IT>
class CItemCompare : public CItemSubtypeFT<Item_func, FT> {
    virtual RewritePlan *
    do_gather_type(const Item_func &i, Analysis &a) const
    {
        LOG(cdb_v) << "CItemCompare (L1139) do_gather func " << i;

        std::string why;

        std::function<EncSet ()> getEncSet =
            [&why, &i] ()
            {
                if (FT == Item_func::FuncType::EQ_FUNC ||
                    FT == Item_func::FuncType::EQUAL_FUNC ||
                    FT == Item_func::FuncType::NE_FUNC) {
                    why = "compare equality";

                    Item *const *const args = i.arguments();
```

```

        if (!args[0]->const_item() && !args[1]->const_item()) {
            why = why + "; join";
            std::cerr << "join";
            return JOIN_EncSet;    /* lambda */
        } else {
            return EQ_EncSet;      /* lambda */
        }
    } else {
        why = "compare order";
        return ORD_EncSet;        /* lambda */
    }
};

const EncSet my_es = getEncSet();

```

里面有一个很可疑的变量：`EncSet`，其中它的子类定义是这样的：

```

const EncSet EQ_EncSet = {
    {
        {oPLAIN, LevelFieldPair(SECLEVEL::PLAINVAL, NULL)},
        {oDET,   LevelFieldPair(SECLEVEL::DET, NULL)},
        {oOPE,   LevelFieldPair(SECLEVEL::OPE, NULL)},
    }
};

const EncSet JOIN_EncSet = {
    {
        {oPLAIN, LevelFieldPair(SECLEVEL::PLAINVAL, NULL)},
        {oDET,   LevelFieldPair(SECLEVEL::DETJOIN, NULL)},
    }
};

const EncSet ORD_EncSet = {
    {
        {oPLAIN, LevelFieldPair(SECLEVEL::PLAINVAL, NULL)},
        {oOPE,   LevelFieldPair(SECLEVEL::OPE, NULL)},
    }
};

const EncSet PLAIN_EncSet = {

```

```
{
    {oPLAIN, LevelFieldPair(SECLEVEL::PLAINVAL, NULL)},
}
};
```

因为我们 where 从句（如果加密的话）需要先从洋葱层剥离出来，因此会先判断加密了哪些洋葱层，这需要用 EncSet 这个类来判断。只需要将明文传输的列的加密方式设置成 PLAIN_EncSet，但是我们并不知道列的名字是什么。

注意到 Item_Func 类中出现了一个数组叫做 args，经 debug 发现 args[0] 和 args[1] 分别存储了 where 中的列名和对应的值。比如说：

```
SELECT *
FROM TEST
WHERE id = 2
```

其中有：args[0] = id; args[1] = 2。

所以只需要利用：

```
strcmp(args[0], "enc_") == 0;
```

来判断就行了。

```
template<Item_func::FuncType FT, class IT>
class CItemCompare : public CItemSubtypeFT<Item_func, FT> {
    virtual RewritePlan *
    do_gather_type(const Item_func &i, Analysis &a) const
{
    Item *const *const args = i.arguments();
    const std::string name = args[0]->name;
    const std::string nenc = "nenc_";
    std::string identifier = name.substr(0, 5);

    if (identifier.compare(nenc) == 0) {
        return typical_gather(a, i, PLAIN_EncSet, why, false, PLAIN_EncSet);
    } else {
        return typical_gather(a, i, es, why, false, PLAIN_EncSet);
    }
}
}
```

```
//In typical_gather:
static RewritePlan *
typical_gather(Analysis &a, const Item_func &i, const EncSet &my_es,
               const std::string &why,
               bool encset_from_intersection,
               const EncSet &other_encset = PLAIN_EncSet)
{
    /*...*/
    if (identifier.compare(nenc) == 0) {
        return new RewritePlanOneOLK(PLAIN_EncSet,
                                     solution.chooseOne(), childr_rp,
                                     rsn);
    } else {
        return new RewritePlanOneOLK(out_es,
                                     solution.chooseOne(), childr_rp,
                                     rsn);
    }
}
```

但是其实这样会有一个问题，args[0]其实是加密后的列名，无法区分到底是不是加密列，为此，指定非加密列前缀为：“nenc_”(non-encryption)

```
select `xinan`.`table_YWQEYPKPTG`.`URXYOCVITKoDET`,`xinan`.`table_YWQEYPKPTG`.`nenc_id` from `xinan`.`table_YWQEYPKPTG` where (`xinan`.`table_YWQEYPKPTG`.`nenc_id` = 2)
```

```
select
`xinan`.`table_YWQEYPKPTG`.`URXYOCVITKoDET`,`xinan`.`table_YWQEYPKPTG`.`nenc_id`
` from `xinan`.`table_YWQEYPKPTG` where
(`xinan`.`table_YWQEYPKPTG`.`URXYOCVITKoDET` = 8770256090147313646 #加密版本
```

```
delete from `xinan`.`table_YWQEYPKPTG` where (`xinan`.`table_YWQEYPKPTG`.`nenc_id` = 3)
```

问题

```
zy@ubuntu: ~  
Current database: xinan  
  
+-----+-----+  
| enc_id | nenc_id |  
+-----+-----+  
| 4      | -3141034267527693959 |  
+-----+-----+  
1 row in set (2.01 sec)  
  
mysql> select * from nima where nenc_id = 3;  
Empty set (0.06 sec)  
  
mysql> insert into nima values(333, 222);  
Query OK, 1 row affected (0.09 sec)  
  
mysql> select * from nima where nenc_id = 222;  
+-----+-----+  
| enc_id | nenc_id |  
+-----+-----+  
| 333    | -5238577302476373302 |  
+-----+-----+  
1 row in set (0.01 sec)  
  
mysql>
```

虽然可以明文 `where` 了，但是 `select` 出来的是一堆乱码（已解决*），加密列正常。（有可能我弄错了）

Update

由于已经实现了 `where` 的部分，因此我们只需要该 `set` 部分的内容即可。经调试，最终 `update` 语句会定位到 `doPairRewrite` 函数中进行重写。

```
void doPairRewrite(FieldMeta& fm, const EncSet& es, const Item_field&  
field_item, const Item& value_item, List<Item>* const res_fields, List<Item>*  
const res_values, Analysis& a)  
{  
  
    /*...*/
```

```

std::string identifier = fm.fname.substr(0, 5);
const std::string nenc = "nenc_";
if (identifier.compare(nenc) == 0) {
    Item* const re_value = const_cast<Item*>(&value_item);
    res_values->push_back(re_value);
} else {
    Item* const re_value = itemTypes.do_rewrite(value_item, olk, *value_rp,
a);
    res_values->push_back(value_item);
}
/*...*/
}

```

The screenshot shows a terminal window with the following MySQL commands and output:

```

mysql> select * from nima;
+-----+-----+
| enc_id | nenc_id |
+-----+-----+
| 1      | 443706402672299823 |
| 3      | 1685858611014138802 |
| 5      | -6652154369429313474 |
| 11     | -7826944420514124195 |
+-----+-----+
4 rows in set (0.04 sec)

mysql> update nima set enc_id = 10 where nenc_id = 101;
Query OK, 1 row affected (0.01 sec)

mysql> select * from nima;
+-----+-----+
| enc_id | nenc_id |
+-----+-----+
| 1      | 443706402672299823 |
| 3      | 1685858611014138802 |
| 5      | -6652154369429313474 |
| 10     | -7826944420514124195 |
+-----+-----+

```

The IDE background shows a project structure on the left and a code outline on the right, including methods like `invalidateConstFieldMeta`, `determineUpdateType`, `doPairRewrite`, `addSalt`, `handleUpdateType`, and `buildDMLDispatcher`.

```
zy@ubuntu: ~  
+-----+-----+  
| enc_id | nenc_id |  
+-----+-----+  
| 1      | 443706402672299823 |  
| 3      | 1685858611014138802 |  
| 5      | -6652154369429313474 |  
| 10     | -7826944420514124195 |  
+-----+-----+  
4 rows in set (0.00 sec)  
  
mysql> update nima set nenc_id = 66 where nenc_id = 101;  
Query OK, 1 row affected (0.08 sec)  
  
mysql> select * from nima;  
+-----+-----+  
| enc_id | nenc_id |  
+-----+-----+  
| 1      | 443706402672299823 |  
| 3      | 1685858611014138802 |  
| 5      | -6652154369429313474 |  
| 10     | -6566244850494412828 |  
+-----+-----+  
4 rows in set (0.00 sec)  
  
mysql>
```

```
in ProxyState getConn  
rewritten query[1]: update (`xinan`.`table_FJLNEHPSCX`) set (`xinan`.`table_FJLNEHPSCX`.`nenc_id`=66, `xinan`.`table_FJLNEHPSCX`.`nenc_id`=66, `xinan`.`table_FJLNEHPSCX`.`nenc_id`=66, `xinan`.`table_FJLNEHPSCX`.`nenc_id`=66 where (`xinan`.`table_FJLNEHPSCX`.`nenc_id` = 101))
```

主要有一点，`set` 会重复执行，虽然一直都是一样的内容，但是很不舒服。

乱码解决方案

我们执行 `insert` 的时候，仅仅是把未加密的内容放进了数据库，但是解密的地方没有改，所以代理服务器就用洋葱解密，解密得到一堆乱码，只需要在解密函数中判断列名是否是加密列即可。

定位到函数 `Rewriter::decryptResults`

```
/*...*/  
  
const std::string nenc = "nenc_";  
unsigned int col_index = 0;
```

```
for (unsigned int c = 0; c < cols; c++) {  
    std::string identifier = res.names[c].substr(0, 5);  
    if (!fm || identifier.compare(nenc) == 0 || dbres.rows[r][c]->is_null()) {  
        res.rows[c][col_index] = dbres.rows[r][c]; //不加密  
    }  
}  
  
/*...*/
```