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;

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);

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();

TEST\_BadItemArgumentCount(i.type(), 2, i.argument\_count());

if (identifier.compare(nenc) == 0) {

return typical\_gather(a, i, PLAIN\_EncSet, why, false, PLAIN\_EncSet);

} else {

return typical\_gather(a, i, my\_es, why, false, PLAIN\_EncSet);

}

}

这个在rewrite\_func.cc : CItemCompare::do\_gather\_type。