Precompiled Catalog Access Query

Ruogu Du, Tianxiong Wang, Ao Zeng

What We Have Done

- 75% Goal:
  - Table catalog sequential access with LLVM compilation and cached plan.
- 100% Goal:
  - Support get object for the rest kinds of catalog
- ~~125% Goal:~~
  - ~~Support index scan~~
- 110% Goal:
  - Rebase our code onto the schema catalog refactoring
  - Add compilation support for insert, update and delete
- 125% Goal:
  - Fix (many) bugs in the existing code base to ensure correctness

What We Have Done

Support compiled sequential scan for fast catalog object lookups, insert, update and delete.

Also cache the compiled query for fast repeat execution.

Supported catalogs includes:

- DatabaseCatalog
- SchemaCatalog
- TableCatalog
- ColumnCatalog
- DatabaseMetricsCatalog
- IndexMetricsCatalog
- IndexCatalog
- LanguageCatalog
- ProcCatalog
- QueryHistoryCatalog
- QueryMetricsCatalog
- SettingCatalog
- TriggerCatalog
- ZoneMapCatalog
- ColumnStatsCatalog
- TableMatricsCatalog

Support Insertion/Deletion

Create insertion/deletion plans and perform binding

Check the plan in the query cache. Compile the plan if cache miss

Execute the compiled insertion/deletion plans

Create Insertion Plans

```
std::vector<std::vector<ExpressionPtr>> tuples;
auto val0 = type::ValueFactory::GetIntegerValue(database_oid);
auto val1 = type::ValueFactory::GetVarcharValue(database_name, nullptr);

auto constant_expr_0 = new expression::ConstantValueExpression(
    val0);
auto constant_expr_1 = new expression::ConstantValueExpression(
    val1);

tuples.emplace_back();
auto &values = tuples[0];
values.emplace_back(ExpressionPtr(constant_expr_0));
values.emplace_back(ExpressionPtr(constant_expr_1));

return InsertTupleWithCompiledPlan(&tuples, txn);
```

Create Deletion Plans

```
// cache miss, get from pg_database
std::vector<oid_t> column_ids(all_column_ids);

auto *db_oid_expr =
    new expression::TupleValueExpression(type::TypeId::INTEGER, 0,
                                         ColumnId::DATABASE_OID);

db_oid_expr->SetBoundOid(catalog_table_->GetDatabaseOid(),
                         catalog_table_->GetOid(),
                         ColumnId::DATABASE_OID);

expression::AbstractExpression *db_oid_const_expr =
    expression::ExpressionUtil::ConstantValueFactory(
        type::ValueFactory::GetIntegerValue(database_oid).Copy());
expression::AbstractExpression *db_oid_equality_expr =
    expression::ExpressionUtil::ComparisonFactory(
        ExpressionType::COMPARE_EQUAL, db_oid_expr, db_oid_const_expr);

expression::AbstractExpression *predicate = db_oid_equality_expr;

// evict cache
txn->catalog_cache.EvictDatabaseObject(database_oid);

return DeleteWithCompiledSeqScan(column_ids, predicate, txn);
```

What We Decided Not to Do

We still decide to pass the wrapped tuple to the catalog object constructor and copy the value out instead of passing a pointer to the constructor. Since the catalog object can serve as a value cache for future references while if we pass the pointer to the wrapped tuple, we need to convert the value to the right type for every get method.

```
TableCatalogObject::TableCatalogObject(codegen::WrappedTuple &wrapped_tuple,
                                       concurrency::TransactionContext *txn)
    : table_oid(wrapped_tuple.GetValue(TableCatalog::ColumnId::TABLE_OID)
                    .GetAs<oid_t>()),
      table_name(wrapped_tuple.GetValue(TableCatalog::ColumnId::TABLE_NAME)
                     .ToString()),
      schema_name(wrapped_tuple.GetValue(TableCatalog::ColumnId::SCHEMA_NAME)
                      .ToString()),
      database_oid(wrapped_tuple.GetValue(TableCatalog::ColumnId::DATABASE_OID)
                       .GetAs<oid_t>()),
      version_id(wrapped_tuple.GetValue(TableCatalog::ColumnId::VERSION_ID)
                     .GetAs<uint32_t>()),
      index_objects(),
      index_names(),
      valid_index_objects(false),
      column_objects(),
      column_names(),
      valid_column_objects(false),
      txn(txn) {}
```

Bugs We Discovered until the Midterm

QueryCache Bug #1298

Open **nwang57** opened this issue 2 days ago · 0 comments

Solved by binding predicates to columns.

Bug in scan #1294

Open **Onmysofa** opened this issue 3 days ago · 0 comments

Solved by reordering the columns in the seq scan. We tried to modify the binding but this caused other problems :(

Bugs We Discovered until the Midterm

Trigger test assumes order in table's trigger list #1293

Open Onmysofa opened this issue 3 days ago · 1 comment

Solved by modifying the test.

More Bugs

- [x] ZoneMap bug when using cached compiled query #1362

  Open Zeninma opened this issue 4 hours ago · 0 comments

- [x] Destruction of uninitialized Value in FillPredicateArray #1363

  Open Zeninma opened this issue 3 hours ago · 0 comments

Solved by passing the expression pointer directly to the ShouldScanZoneMap method instead of using PredicateInfo.

More Bugs

- [x] Assertion fail when insert and delete in the same transaction #1336

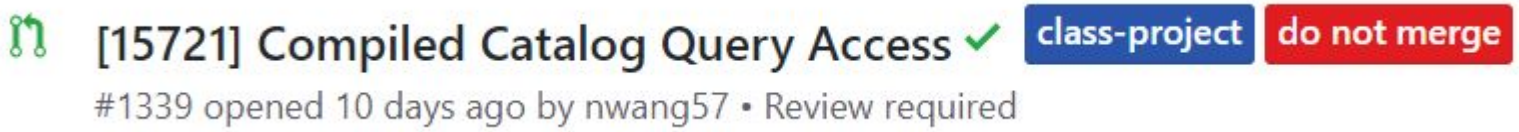
Kind-hearted Prashanth solved this by completing the implementation of delete.

Something Else

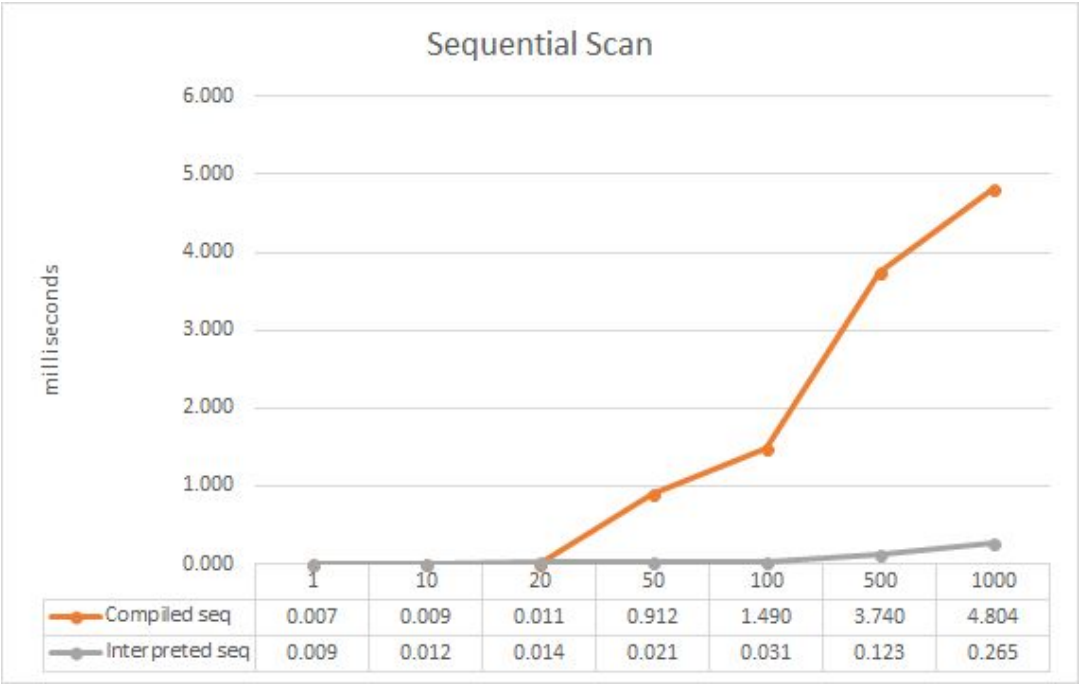
```
engine_.reset(llvm::EngineBuilder(std::move(m))
                    .setEngineKind(llvm::EngineKind::JIT)
                    .setMCJITMemoryManager(
                        llvm::make_unique<PelotonMM>(function_symbols_))
                    .setMCPU(llvm::sys::getHostCPUName())
                    .setErrorStr(&err_str_)
                    .setOptLevel(llvm::CodeGenOpt::Level::Aggressive)
                    .create());
```

Setting optimization level can slight promote the performance

Evaluation

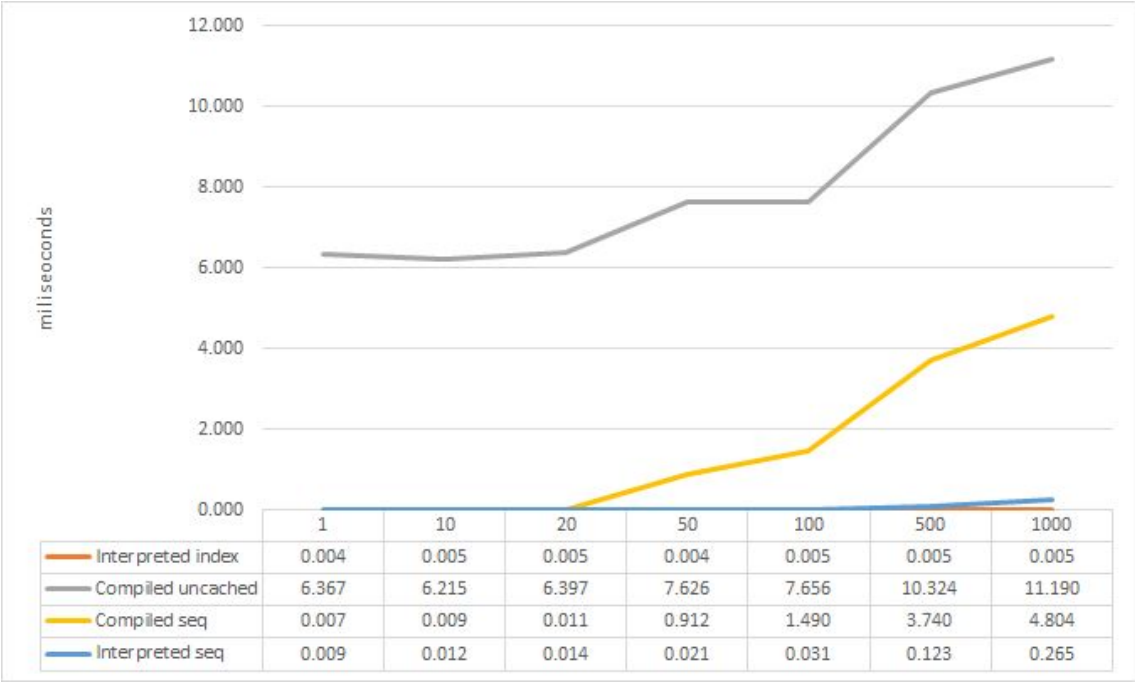

[15721] Compiled Catalog Query Access
class-project
do not merge
#1339 opened 10 days ago by nwang57 • Review required

Evaluation



	1	10	20	50	100	500	1000
Compiled seq	0.007	0.009	0.011	0.912	1.490	3.740	4.804
Interpreted seq	0.009	0.012	0.014	0.021	0.031	0.123	0.265

Evaluation



	1	10	20	50	100	500	1000
Interpreted index	0.004	0.005	0.005	0.004	0.005	0.005	0.005
Compiled uncached	6.367	6.215	6.397	7.626	7.656	10.324	11.190
Compiled seq	0.007	0.009	0.011	0.912	1.490	3.740	4.804
Interpreted seq	0.009	0.012	0.014	0.021	0.031	0.123	0.265

Future Works

- Support Compiled IndexScan
- LLVM passes for compilation optimization

Thanks!

thank you
danke
謝謝
ngiyabonga
teşekkür ederim
gracias
рахмат
Баярлалаа
спасибо
mersi
kia ora
barka
welalin
tack
spas
dank je
misaotra
matondo
paldies
grazzi
mahalo
tapadh leat
хвала
asante
manana
obrigada
tenki
faafetai lava
vinaka
blagodaram
спасибі
kiitos
dankie
dhanyavad
nanni
nandri
mauruuru
köszönöm
hvala
gracie
bayarlalaa
bedankt
enkosi
dziękuję
djiere dieuf
tau
mochchakkeram
murakoze
мамнун
дякую
chokrane
chnorakaloutioun
grafias ago
gràcies
sulpay
go raibh maith agat
sobodi
děkuji
sağolun
sukriya
kop khun krap
taiku
arigatô
takk
dakujem
trugarez
obrigado
mèsi
ありがとう
tanemirt
grazie
najis tuke
didi madloba
kam sah hamnida
rahmat
terima kasih
rahmet
diolch
dhanyavadagalu
shukriya
merce
мерси
xièxie
ευχαριστώ
merci
감사합니다