LLVM Query Runtime

VALKyrie

Arindam
Kaushik

Ladan
Vinayak

What we did last time

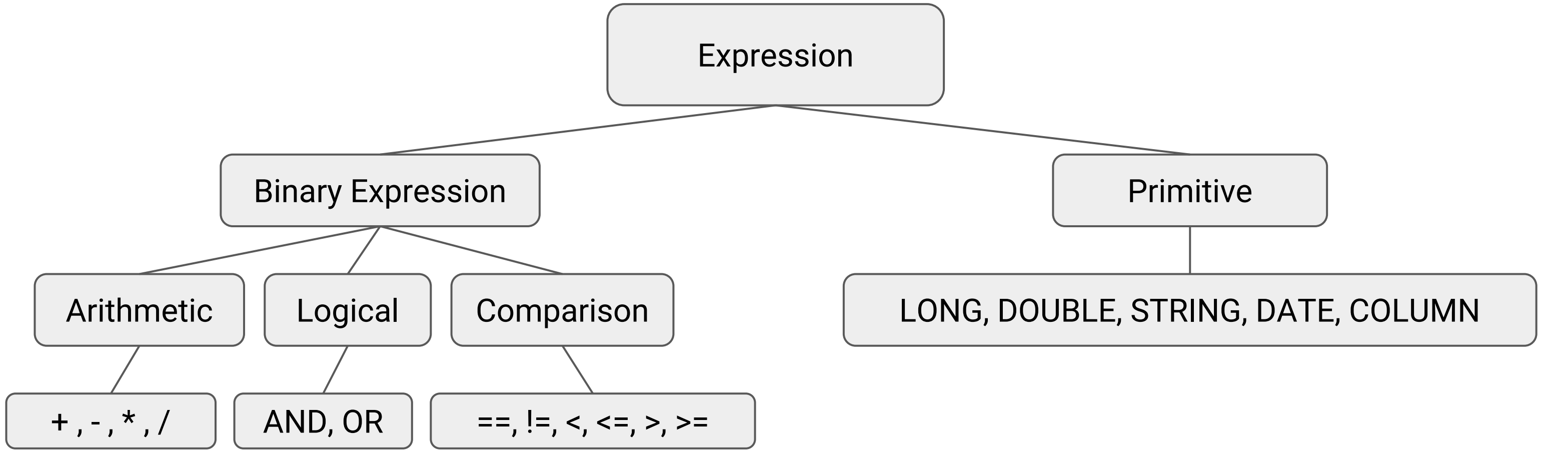
Scan operator

```
valkyrie> SELECT * FROM <table_name>;
```

This week

1. Expression tree
2. Selection operator

Expression tree



getValue()

1. Every expression has member `getValue()`
2. `getValue()` returns a `Value*`
   [LLVM omnivorous class representing a code element]
3. While doing selection, just call `getValue()` on the top level expression
4. This will return a boolean on which we do a conditional branch

Example -

```
Value* AdditionExpression::getValue() {
   IRBuilder<>* builder = codegen::getBuilder();
   return builder->CreateAdd(leftExpression->getValue(), rightExpression->getValue());
}
```

More examples

```
Value* AndExpression::getValue() {
   IRBuilder<>* builder = codegen::getBuilder();
   return builder->CreateAnd(leftExpression->getValue(), rightExpression->getValue());
}

Value* EqualExpression::getValue() {
   IRBuilder<>* builder = codegen::getBuilder();
   switch(rightExpression->getType()) {
       case LONGVALUEEXPRESSION:
           return builder->CreateICmpEQ(leftExpression->getValue(), rightExpression->getValue());
       case DOUBLEVALUEEXPRESSION:
           return builder->CreateFCmpOEQ(leftExpression->getValue(), rightExpression->getValue());
       default:
           cerr << "Unknown expression type!" << endl;
           exit(-1);
   }
   return NULL;
}
```

Column Primitive

```
Value* ColExpression::getValue() {
   IRBuilder<>* builder = codegen::getBuilder();
   std::size_t pos = codegen::getAttPos(colname);
   DataType dt = codegen::getAttType(colname);
   Value* tupleptr = codegen::getTupleptr();

   Value *indices[1];
   indices[0] = ConstantInt::get(Type::getInt32Ty(getGlobalContext()), pos);
   ArrayRef<Value*> indicesRef(indices);
   Value *data = builder->CreateLoad(builder->CreateInBoundsGEP(tupleptr, indicesRef));
   switch(dt){
       case LONG:
           return data;
       case DOUBLE:
           return builder->CreateUIToFP(data, Type::getDoubleTy(getGlobalContext()));
   }
   return NULL;
}
```

Selection Operator

```
void codegen::selectConsume(Expression *clause, valkyrie::Operator *parent){
   Value* condV = clause->getValue();
   BasicBlock *cond_true = BasicBlock::Create(context, "if"+to_string(nameCtr++), mainFunction);

   //If the tuple does not satisfy the where condition and also after returning from parent’s produce
   BasicBlock *merge = BasicBlock::Create(context, "continue"+to_string(nameCtr++), mainFunction);
   builder->CreateCondBr(condV, cond_true, merge);
   builder->SetInsertPoint(cond_true);
   if(parent != NULL){
       parent->consume();
   }
   builder->CreateBr(merge);
   builder->SetInsertPoint(merge);
}
```

Demo

Next steps

1. Handle strings in the Expression Tree [Pointer arithmetic or ConstantDataArray]
2. Schema mappings for projection
3. Projection and materialization