Parsing SQL like syntax, design pattern

I am trying mock sql syntax to build a simple sql like interface to a key-value storage. The values are essentially POJOs

An example would be

```
select A.B.C from OBJ_POOL where A.B.X = 45 AND A.B.Y > '88' AND A.B.Z != 'abc';
```

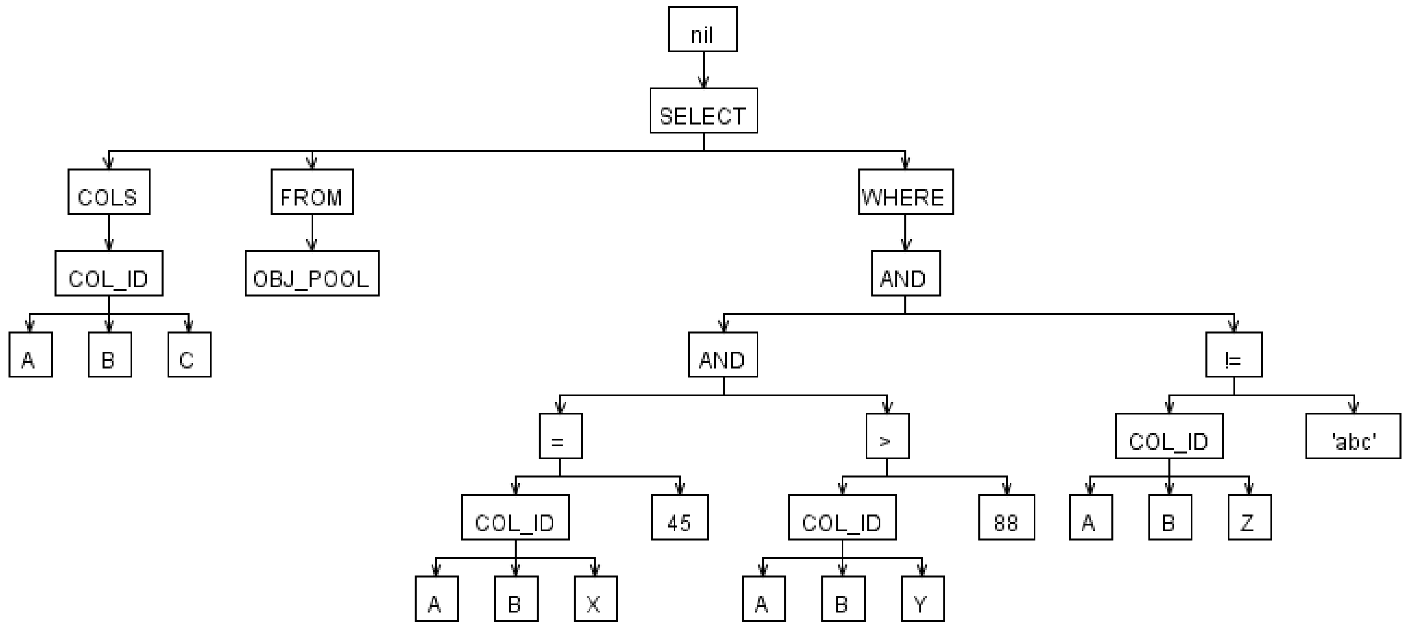
OBJ_POOL is just a list of POJOs of the same class. In this example A would be the base class.

```
Class A
    Class B
        String C
        Integer X
        String Y
        String Z
```

Now A.B.C is equivalent A.getB().getC()

I am using Antlr to parse the above statement to get an AST, and then `hoping` to use Apache BeanUtils to reflectively get/set the field names.

I wrote the grammar thats builds an AST



Now I am facing two problems

1. How should the visitor be implemented for the where clause ? A.B.X = 45 implies all objects having field X as 45, how should the filtering happen is there any nice way to do this ?
2. Is there any way to traverse the generated AST without cluttering the visitor code with custom logic (storage access,property getters/setters etc..)

The second problem is more worrying since there might be many things that the statement might do.

In a nutshell any suggestions/links/design-patterns to nicely parse a small subset of the sql select statment would be greatly appreciated

Thanks

java sql design-patterns antlr abstract-syntax-tree

asked Apr 30 '12 at 7:26

jack_carver
916 1 9 24

1 I got some example that I post before. Hope will help you... – Crazenezz Apr 30 '12 at 8:28

1 Answer

You can do this much like how I demonstrated in my blog posts (and since I know you read those, I won't go in much detail). The only difference in this case is that each of your rows of data has its own scope. An easy way to pass this scope along is by providing it as a parameter to the `eval(...)` method.

Below is a quick demo of how this could be implemented. Note that I quickly hacked this together based on my blog posts: not all functionality is available (see the many `TODO`'s, and there are likely (small) bugs in it as well. Use at your own risk!).

Besides ANTLR v3.3, you need the following 3 files for this demo:

Select.g

```
grammar Select;

options {
  output=AST;
}

tokens {
  // imaginary tokens
  ROOT;
  ATTR_LIST;
  UNARY_MINUS;

  // literal tokens
  Eq     = '=';
  NEq    = '!=';
  LT     = '<';
  LTEq   = '<=';
  GT     = '>';
  GTEq   = '>=';
  Minus  = '-';
  Not    = '!';
  Select = 'select';
  From   = 'from';
  Where  = 'where';
  And    = 'AND';
  Or     = 'OR';
}

parse
 : select_stat EOF -> ^(ROOT select_stat)
 ;

select_stat
 : Select attr_list From Id where_stat ';' -> ^(Select attr_list Id where_stat)
 ;

attr_list
 : Id (',' Id)* -> ^(ATTR_LIST Id+)
 ;

where_stat
 : Where expr -> expr
 |            -> ^(Eq Int["1"] Int["1"])
                 // no 'where', insert '1=1' which is always true
 ;

expr
 : or_expr
 ;

or_expr
 : and_expr (Or^ and_expr)*
 ;

and_expr
 : eq_expr (And^ eq_expr)*
 ;

eq_expr
 : rel_expr ((Eq | NEq)^ rel_expr)*
 ;

rel_expr
 : unary_expr ((LT | LTEq | GT | GTEq)^ unary_expr)?
 ;

unary_expr
 : Minus atom -> ^(UNARY_MINUS atom)
 | Not atom   -> ^(Not atom)
 | atom
 ;

atom
 : Str
 | Int
```

```
  | Id
  | '(' expr ')' -> expr
  ;

Id    : ('a'..'z' | 'A'..'Z' | '_') ('a'..'z' | 'A'..'Z' | '_' | Digit)*;
Str   : '\'' ('\'\'' | ~('\'' | '\r' | '\n'))* '\''
        {
         // strip the surrounding quotes and replace '' with '
         setText($text.substring(1, $text.length() - 1).replace("''", "'"));
        }
      ;
Int   : Digit+;
Space : (' ' | '\t' | '\r' | '\n') {skip();};

fragment Digit : '0'..'9';
```

SelectWalker.g

```
tree grammar SelectWalker;

options {
  tokenVocab=Select;
  ASTLabelType=CommonTree;
}

@header {
  import java.util.List;
  import java.util.Map;
  import java.util.Set;
}

@members {

  private Map<String, List<B>> dataPool;

  public SelectWalker(CommonTreeNodeStream nodes, Map<String, List<B>> data) {
    super(nodes);
    dataPool = data;
  }
}

query returns [List<List<Object>> result]
 : ^(ROOT select_stat) {$result =
(List<List<Object>>)$select_stat.node.eval(null);}
 ;

select_stat returns [Node node]
 : ^(Select attr_list Id expr)
```

```
    {$node = new SelectNode($attr_list.attributes, dataPool.get($Id.text),
$expr.node);}
 ;

attr_list returns [List<String> attributes]
@init{$attributes = new ArrayList<String>();}
 : ^(ATTR_LIST (Id {$attributes.add($Id.text);})+)
 ;

expr returns [Node node]
 : ^(Or a=expr b=expr)   {$node = null; /* TODO */}
 | ^(And a=expr b=expr)  {$node = new AndNode($a.node, $b.node);}
 | ^(Eq a=expr b=expr)   {$node = new EqNode($a.node, $b.node);}
 | ^(NEq a=expr b=expr)  {$node = new NEqNode($a.node, $b.node);}
 | ^(LT a=expr b=expr)   {$node = null; /* TODO */}
 | ^(LTEq a=expr b=expr) {$node = null; /* TODO */}
 | ^(GT a=expr b=expr)   {$node = new GTNode($a.node, $b.node);}
 | ^(GTEq a=expr b=expr) {$node = null; /* TODO */}
 | ^(UNARY_MINUS a=expr) {$node = null; /* TODO */}
 | ^(Not a=expr)         {$node = null; /* TODO */}
 | Str                   {$node = new AtomNode($Str.text);}
 | Int                   {$node = new AtomNode(Integer.valueOf($Int.text));}
 | Id                    {$node = new IdNode($Id.text);}
 ;
```

Main.java

(yes, stick all these Java classes in the same file: `Main.java` *)*

```java
import org.antlr.runtime.*;
import org.antlr.runtime.tree.*;
import org.antlr.stringtemplate.*;
import java.util.*;

public class Main {

  static Map<String, List<B>> getData() {
    Map<String, List<B>> map = new HashMap<String, List<B>>();
    List<B> data = new ArrayList<B>();
    data.add(new B("id_1", 345, "89", "abd"));
    data.add(new B("id_2", 45, "89", "abd"));
    data.add(new B("id_3", 1, "89", "abd"));
    data.add(new B("id_4", 45, "8", "abd"));
    data.add(new B("id_5", 45, "89", "abc"));
    data.add(new B("id_6", 45, "99", "abC"));
    map.put("poolX", data);
    return map;
  }

  public static void main(String[] args) throws Exception {
    String src = "select C, Y from poolX where X = 45 AND Y > '88' AND Z !=
'abc';";
    SelectLexer lexer = new SelectLexer(new ANTLRStringStream(src));
    SelectParser parser = new SelectParser(new CommonTokenStream(lexer));
    CommonTree tree = (CommonTree)parser.parse().getTree();
    SelectWalker walker = new SelectWalker(new CommonTreeNodeStream(tree),
getData());
    List<List<Object>> result = walker.query();
    for(List<Object> row : result) {
      System.out.println(row);
    }
  }
}

class B {

  String C;
  Integer X;
  String Y;
  String Z;

  B(String c, Integer x, String y, String z) {
    C = c;
    X = x;
    Y = y;
    Z = z;
  }

  Object getAttribute(String attribute) {
    if(attribute.equals("C")) return C;
    if(attribute.equals("X")) return X;
    if(attribute.equals("Y")) return Y;
    if(attribute.equals("Z")) return Z;
    throw new RuntimeException("Unknown attribute: B." + attribute);
    // or use your Apache Bean-util API, or even reflection here instead of the
above...
  }
}

interface Node {
  Object eval(B b);
}

class AtomNode implements Node {

  final Object value;
  AtomNode(Object v) {
    value = v;
  }

  public Object eval(B b) {
    return value;
  }
}

abstract class BinNode implements Node {

  final Node left;
  final Node right;

  BinNode(Node l, Node r) {
    left = l;
    right = r;
  }

  public abstract Object eval(B b);
}

class AndNode extends BinNode {

  AndNode(Node l, Node r) {
    super(l, r);
  }

  @Override
  public Object eval(B b) {
    return (Boolean)super.left.eval(b) && (Boolean)super.right.eval(b);
  }
}

class EqNode extends BinNode {

  EqNode(Node l, Node r) {
    super(l, r);
  }

  @Override
  public Object eval(B b) {
    return super.left.eval(b).equals(super.right.eval(b));
  }
}

class NEqNode extends BinNode {
  NEqNode(Node l, Node r) {
    super(l, r);
  }

  @Override
  public Object eval(B b) {
    return !super.left.eval(b).equals(super.right.eval(b));
  }
}

class GTNode extends BinNode {

  GTNode(Node l, Node r) {
    super(l, r);
  }

  @Override
  public Object eval(B b) {
    return
((Comparable)super.left.eval(b)).compareTo((Comparable)super.right.eval(b)) > 0;
  }
}

class IdNode implements Node {

  final String id;

  IdNode(String i) {
    id = i;
  }

  @Override
  public Object eval(B b) {
    return b.getAttribute(id);
  }
}

class SelectNode implements Node {

  final List<String> attributes;
  final List<B> data;
  final Node expression;

  SelectNode(List<String> a, List<B> d, Node e) {
    attributes = a;
    data = d;
    expression = e;
  }
  @Override
  public Object eval(B ignored) {
    List<List<Object>> result = new ArrayList<List<Object>>();
    for(B b : data) {
      if((Boolean)expression.eval(b)) {
        // 'b' passed, check which attributes to include
        List<Object> row = new ArrayList<Object>();
        for(String attr : attributes) {
          row.add(b.getAttribute(attr));
        }
        result.add(row);
      }
    }
    return result;
  }
}
```

If you now generate the lexer, parser and tree walker and run the Main class:

```
java -cp antlr-3.3.jar org.antlr.Tool Select.g
java -cp antlr-3.3.jar org.antlr.Tool SelectWalker.g
javac -cp antlr-3.3.jar *.java
java -cp .:antlr-3.3.jar Main
```

you will see that the output for the query:

```
select C, Y from poolX where X = 45 AND Y > '88' AND Z != 'abc';
```

with input:

```
C           X       Y       Z
"id_1"      345     "89"    "abd"
"id_2"      45      "89"    "abd"
"id_3"      1       "89"    "abd"
"id_4       45      "8"     "abd"
"id_5"      45      "89"    "abc"
"id_6"      45      "99"    "abC"
```

is:

```
[id_2, 89]
[id_6, 99]
```

And note that if the `where` statement is omitted, the expression `1 = 1` is automatically inserted, causing the query:

```
select C, Y from poolX;
```

to print the following:

```
[id_1, 89]
[id_2, 89]
[id_3, 89]
[id_4, 8]
[id_5, 89]
[id_6, 99]
```

edited Aug 2 '14 at 21:11
Benjamin
14.9k 26 113 220

answered May 1 '12 at 19:13
Bart Kiers
127k 28 230 241

1 This looks neat ... really appreciate the effort. Thanks – jack_carver May 2 '12 at 1:29

1 linked blog (bkiers.blogspot.com/2011/03/…) is not working – myuce May 27 '16 at 11:24