# Working with SQL Syntax trees in .NET

Isak Sky

# About me

Software Engineer working at Xledger, mostly with F# and SQL.

**Open source work:**

github.com/isaksky


xledger.com

1311 Interquest Pkwy

# Agenda

1. What is a syntax tree?
2. What can it be used for?
3. Tutorial / Demos

# What is a Syntax Tree?

Wikipedia:

A tree representation of the **abstract syntactic structure** of **source code** written in a programming language.

# Syntax Tree Example 1 / 3

```
while b ≠ 0
  if a > b
    a := a - b
  else
    b := b - a
return a
```


[Wikipedia]

# Syntax Tree Example 2 / 3

```
public class Person {
  public int age;
  public string name;
  public DateTime birthdate;
}
```

https://roslynquoter.azurewebsites.net/

```
CompilationUnit()
.WithMembers(
    SingletonList<MemberDeclarationSyntax>(
        ClassDeclaration("Person")
        .WithModifiers(
            TokenList(
                Token(SyntaxKind.PublicKeyword)))
        .WithMembers(
            List<MemberDeclarationSyntax>(
                new MemberDeclarationSyntax[]{
                    FieldDeclaration(
                        VariableDeclaration(
                            PredefinedType(
                                Token(SyntaxKind.IntKeyword)))
                        .WithVariables(
                            SingletonSeparatedList<VariableDeclaratorSyntax>(
                                VariableDeclarator(
                                    Identifier("age")))))
                    .WithModifiers(
                        TokenList(
                            Token(SyntaxKind.PublicKeyword))),
                    FieldDeclaration(
                        VariableDeclaration(
                            PredefinedType(
                                Token(SyntaxKind.StringKeyword)))
                        .WithVariables(
                            SingletonSeparatedList<VariableDeclaratorSyntax>(
                                VariableDeclarator(
                                    Identifier("name")))))
                    .WithModifiers(
                        TokenList(
                            Token(SyntaxKind.PublicKeyword))),
                    FieldDeclaration(
                        VariableDeclaration(
                            IdentifierName("DateTime"))
                        .WithVariables(
                            SingletonSeparatedList<VariableDeclaratorSyntax>(
                                VariableDeclarator(
                                    Identifier("birthdate")))))
                    .WithModifiers(
                        TokenList(
                            Token(SyntaxKind.PublicKeyword)))})))
.NormalizeWhitespace()
```

# Syntax Tree Example 3 / 3

```
(loop [x 10]
  (when (> x 1)
    (println x)
    (recur (- x 2))))
```



```
(loop [x 10]
  (when (> x 1)
    (println x)
    (recur (- x 2))))
```

# How could a SQL Syntax tree be used?

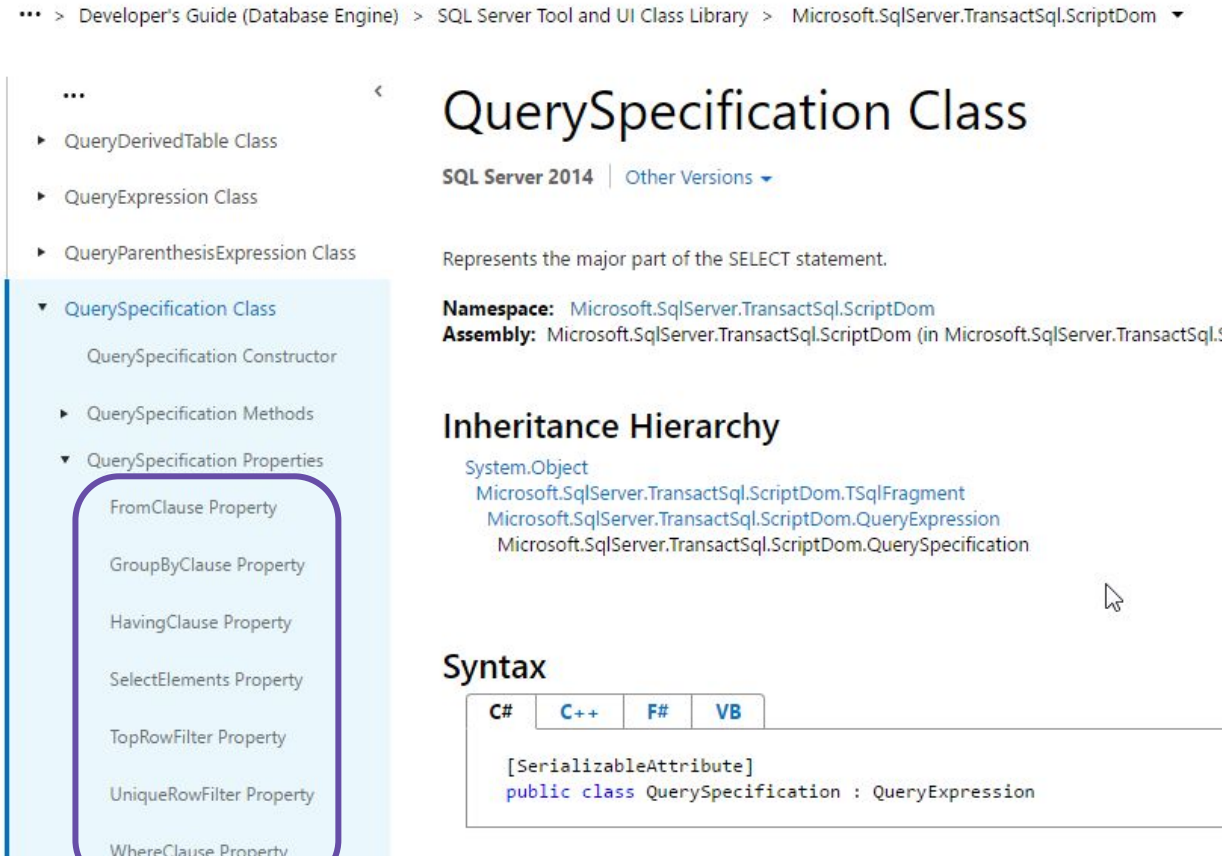
- Analyzing queries
  - Finding performance problems
  - Checking permissions
- Rewriting queries
  - Optimize poorly written queries
  - Add/remove conditions to WHERE clause for security
- Building up SQL Statements dynamically
  - Safer to use than string concatenation, but complete

# SQL Syntax Trees in .NET

Available via
Microsoft.SqlServer.TransactSql.ScriptDom.dll

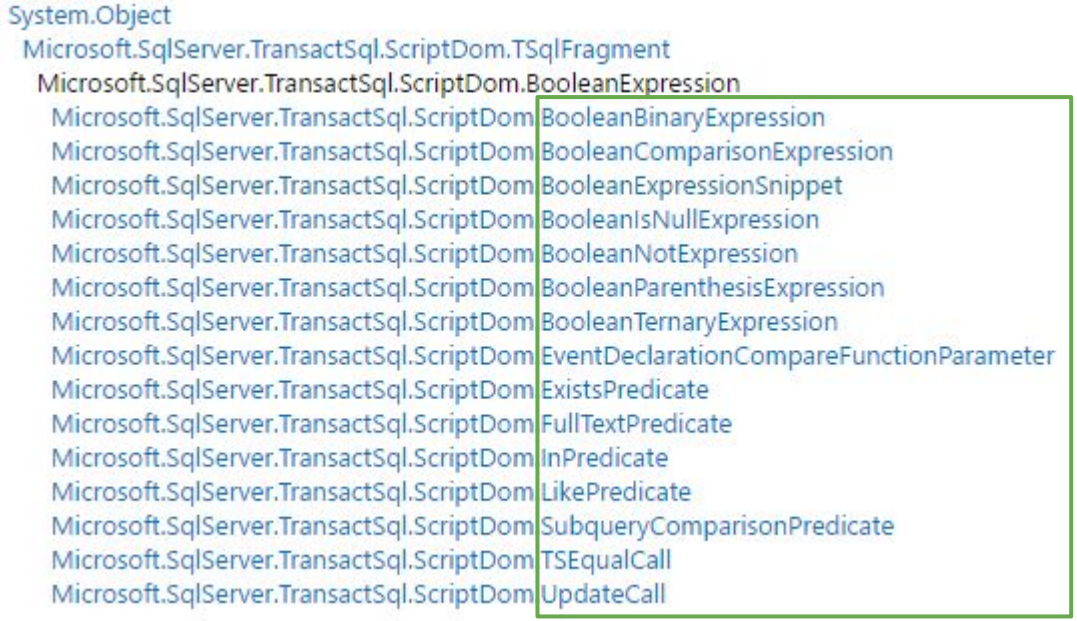
**Available in redistributable packages**

**Based on same grammar as engine ***

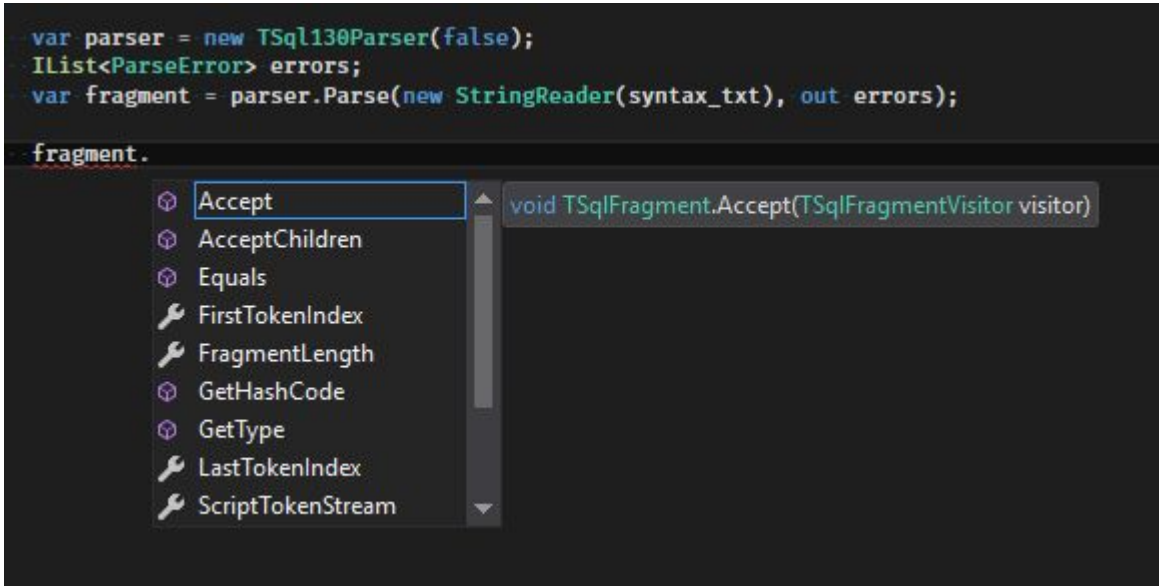


# SQL Syntax Trees

Available via
Microsoft.SqlServer.TransactSql.ScriptDom.dll



# Using the parser



Visitor - works well for simple cases

```
var parser = new TSql130Parser(false);
IList<ParseError> errors;
var fragment = parser.Parse(new StringReader(syntax_txt), out errors);

var analyzer = new MyNaiveAnalyzer();
fragment.Accept(analyzer);
```

Demo

```
class MyNaiveAnalyzer : TSqlFragmentVisitor {
    StringBuilder _sb = new StringBuilder();
    public MyNaiveAnalyzer() { }

    public override void Visit(AddAlterFullTextIndexAction node) {
        base.Visit(node);
    }

    public override void Visit(ColumnReferenceExpression node) {
        var col = String.Join(".", node.MultiPartIdentifier.Identifiers.Select(id => id.Value));
        _sb.AppendFormat("Found column: {0}.\n", col);
    }

    public override void Visit(NamedTableReference node) {
        var table = String.Join(".", node.SchemaObject.Identifiers.Select(id => id.Value));
        _sb.AppendFormat("Found named table: {0}.\n", table);
    }

    public string GetResult() => _sb.ToString();
}
```

# Visitors

- Fine and useful for simple cases
- If you need to keep track of “where you are”:
  - More trouble than it is worth
  - Can lead to verbose, hard to follow code

# Alternative to Visitors?

- C# - Looping, recursion, type checking, casting

```
foreach (TSqlBatch batch in script.Batches)
{
  foreach (TSqlStatement statement in batch.Statements)
  {
    var selectStatement = statement as SelectStatement;
    if (selectStatement != null && selectStatement.QueryExpression != null)
    {
      var querySpecification = selectStatement.QueryExpression as QuerySpecification;
      foreach (TableSource tableSource in querySpecification.FromClauses)
      {
      }
    }
  }
}
```

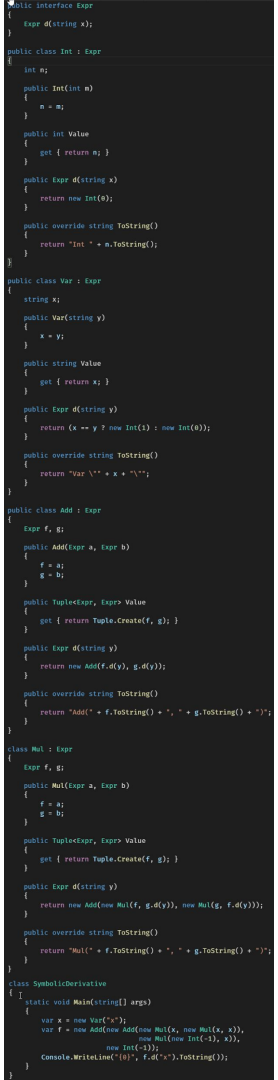
# F# - Sometimes less verbose

Symbolic derivative in F#

```
type Expr =
  | Int of int
  | Var of string
  | Add of Expr * Expr
  | Mul of Expr * Expr

let rec d f x =
  match f with
  | Var y when x=y -> Int 1
  | Int _ | Var _ -> Int 0
  | Add(f, g) -> Add(d f x, d g x)
  | Mul(f, g) -> Add(Mul(f, d g x), Mul(g, d f x))

let f =
  let x = Var "x"
  Add(Add(Mul(x, Mul(x, x)), Mul(Int -1, x)), Int -1)

d f "x"
```

# F# - Sometimes less verbose

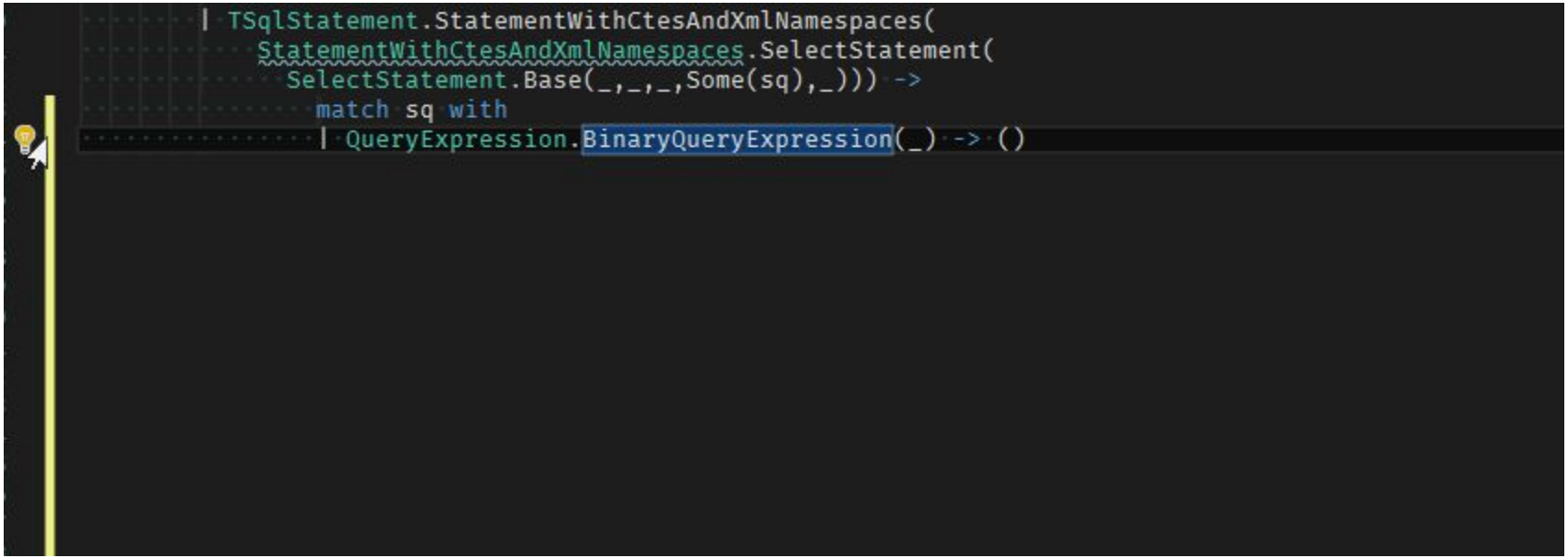
[Symbolic derivative in C#]



```
public interface Expr
{
    Expr d(string x);
}

public class Int : Expr
{
    int n;

    public Int(int m)
    {
        n = m;
    }

    public int Value
    {
        get { return n; }
    }

    public Expr d(string x)
    {
        return new Int(0);
    }

    public override string ToString()
    {
        return "Int " + n.ToString();
    }
}

public class Var : Expr
{
    string x;

    public Var(string y)
    {
        x = y;
    }

    public string Value
    {
        get { return x; }
    }

    public Expr d(string y)
    {
        return (x == y ? new Int(1) : new Int(0));
    }

    public override string ToString()
    {
        return "Var \"" + x + "\"";
    }
}

public class Add : Expr
{
    Expr f, g;

    public Add(Expr a, Expr b)
    {
        f = a;
        g = b;
    }

    public Tuple<Expr, Expr> Value
    {
        get { return Tuple.Create(f, g); }
    }

    public Expr d(string y)
    {
        return new Add(f.d(y), g.d(y));
    }

    public override string ToString()
    {
        return "Add(" + f.ToString() + ", " + g.ToString() + ")";
    }
}

class Mul : Expr
{
    Expr f, g;

    public Mul(Expr a, Expr b)
    {
        f = a;
        g = b;
    }

    public Tuple<Expr, Expr> Value
    {
        get { return Tuple.Create(f, g); }
    }

    public Expr d(string y)
    {
        return new Add(new Mul(f, g.d(y)), new Mul(g, f.d(y)));
    }

    public override string ToString()
    {
        return "Mul(" + f.ToString() + ", " + g.ToString() + ")";
    }
}

class SymbolicDerivative
{
    static void Main(string[] args)
    {
        var x = new Var("x");
        var f = new Add(new Add(new Mul(x, new Mul(x, x)),
                                new Mul(new Int(-1), x)),
                        new Int(-1));
        Console.WriteLine("{0}", f.d("x").ToString());
    }
}
```

# Alternative to Visitors?

- F# - Looping, recursion, pattern matching



# FsSqlDom

An F# Library I wrote.

Utilities to convert back and forth to powerful F# constructs that enable pattern matching.

~14 KLOC Generated F#

**F# Discriminated Union**

```
type Shape =
    | Rectangle of width : float * length : float
    | Circle of radius : float
    | Prism of width : float * float * height : float
```

Represents a closed set of alternatives

# FsSqlDom

Example - F# version of ScriptDom.BooleanExpression:

```
and [<RequireQualifiedAccess>] BooleanExpression = (* IsAbstract = true *)
    | BooleanBinaryExpression of BinaryExpressionType:ScriptDom.BooleanBinaryExpressionType * FirstExpression:B
    | BooleanComparisonExpression of ComparisonType:ScriptDom.BooleanComparisonType * FirstExpression:ScalarExp
    | BooleanExpressionSnippet of Script:String option
    | BooleanIsNullExpression of Expression:ScalarExpression option * IsNot:bool
    | BooleanNotExpression of Expression:BooleanExpression option
    | BooleanParenthesisExpression of Expression:BooleanExpression option
    | BooleanTernaryExpression of FirstExpression:ScalarExpression option * SecondExpression:ScalarExpression o
    | EventDeclarationCompareFunctionParameter of EventValue:ScalarExpression option * Name:EventSessionObjectN
    | ExistsPredicate of Subquery:ScalarSubquery option
    | FullTextPredicate of Columns:(ColumnReferenceExpression) list * FullTextFunctionType:ScriptDom.FullTextFu
    | InPredicate of Expression:ScalarExpression option * NotDefined:bool * Subquery:ScalarSubquery option * Va
    | LikePredicate of EscapeExpression:ScalarExpression option * FirstExpression:ScalarExpression option * Not
    | SubqueryComparisonPredicate of ComparisonType:ScriptDom.BooleanComparisonType * Expression:ScalarExpressi
    | TSEqualCall of FirstExpression:ScalarExpression option * SecondExpression:ScalarExpression option
    | UpdateCall of Identifier:Identifier option
```

# Demo

- Table relationship visualizer
  - Strategy
    - For each {**`view`**, **`table valued function`**, **`stored proc`**}
      - Look at FROM clause,
      - Analyze, find table joins
  - Solution written in F#, spits out Javascript + HTML for rendering
- Syntax Builder
  - How does one build up SQL Statements dynamically?

# Links + Q&A

- My blog post “**Working with SQL syntax trees in F#**”
  - https://goo.gl/nNBbLM
- Microsoft.SqlServer.TransactSql.ScriptDom
  - https://msdn.microsoft.com/en-us/library/microsoft.sqlserver.transactsql.scriptdom.aspx
- FsSqlDom - ScriptDom remix for F#
  - github.com/isaksky/FsSqlDom
- T-SQL Swiss Knife using the ScriptDom T-SQL Parser by Arvind Shyamsundar
  - https://www.youtube.com/watch?v=CciVxRFXgH8