That's me this past Saturday enjoying dinner on a rare cool evening in Houston on our way to the symphony.

I like to explore interesting data topics.

Ready to go down the M rabbit hole?

#mondayisforM

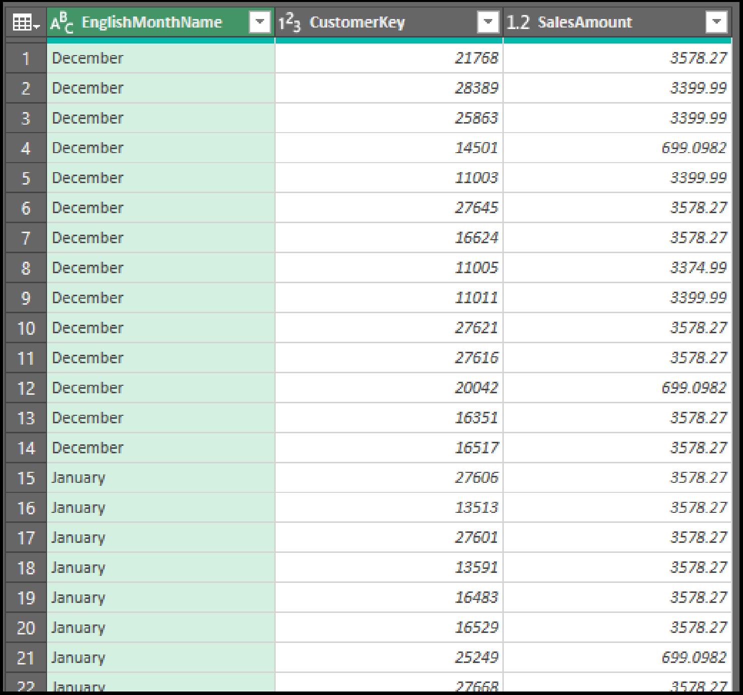
If you want to talk about Power Query, SQL, Excel or other data topics, let's connect. You can follow me or this hashtag for more posts like this.

flexyourdata.com | youtube.com/@flexyourdata | linkedin.com/in/owenhprice

# Power Query (M): Use Value.Expression to view SQL text and optimize query folding

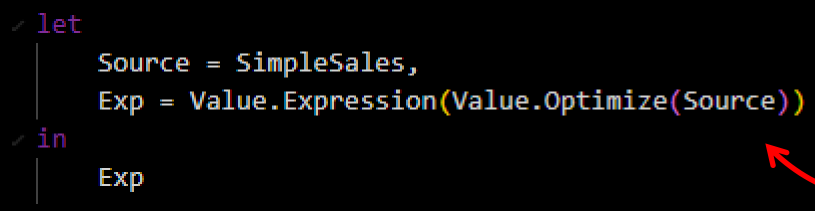
## Suppose we a query against the Adventure Works DW 2019 database called SimpleSales

```
let
    Source = Sql.Database(
        "localhost
        "AdventureWorksDW2019",
        [
            Query=  "SELECT d.EnglishMonthName,
                            fis.CustomerKey,
                            fis.SalesAmount
                     FROM FactInternetSales fis
                         INNER JOIN DimDate d
                             ON fis.OrderDateKey = d.DateKey;",
            CreateNavigationProperties=false,
            HierarchicalNavigation=true
        ])
in
    Source
```



|  | EnglishMonthName | CustomerKey | SalesAmount |
|---|---|---|---|
| 1 | December | 21768 | 3578.27 |
| 2 | December | 28389 | 3399.99 |
| 3 | December | 25863 | 3399.99 |
| 4 | December | 14501 | 699.0982 |
| 5 | December | 11003 | 3399.99 |
| 6 | December | 27645 | 3578.27 |
| 7 | December | 16624 | 3578.27 |
| 8 | December | 11005 | 3374.99 |
| 9 | December | 11011 | 3399.99 |
| 10 | December | 27621 | 3578.27 |
| 11 | December | 27616 | 3578.27 |
| 12 | December | 20042 | 699.0982 |
| 13 | December | 16351 | 3578.27 |
| 14 | December | 16517 | 3578.27 |
| 15 | January | 27606 | 3578.27 |
| 16 | January | 13513 | 3578.27 |
| 17 | January | 27601 | 3578.27 |
| 18 | January | 13591 | 3578.27 |
| 19 | January | 16483 | 3578.27 |
| 20 | January | 16529 | 3578.27 |
| 21 | January | 25249 | 699.0982 |
| 22 | January | 27668 | 3578.27 |

# Value.Expression returns an *abstract syntax tree* (AST)



```
let
    Source = SimpleSales,
    Exp = Value.Expression(Value.Optimize(Source))
in
    Exp
```

Value.Optimize attempts to optimize the query in its parameter. If it can be optimized, the optimized expression is passed to Value.Expression. If it can't, the expression is passed directly to Value.Expression

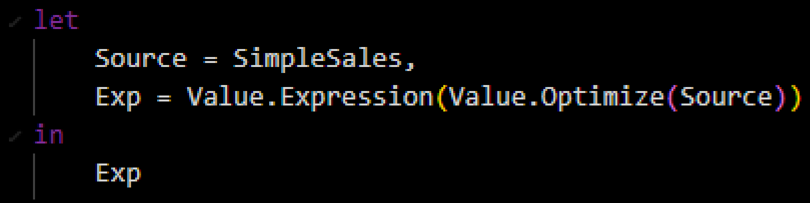
Value.Expression returns an *abstract syntax tree* (AST) for the value's expression.

This is a navigable representation of the code that is producing the value.

The AST for an expression can be highly complex, depending on the complexity of the query that produces the value.

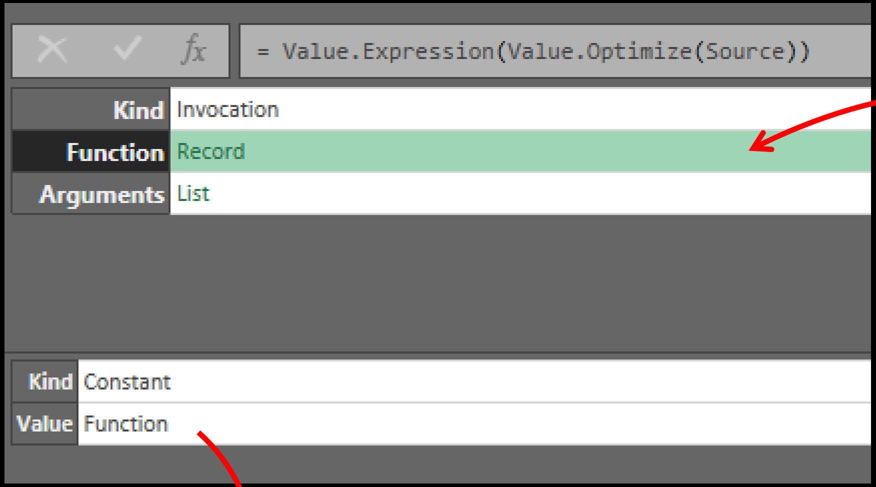
For SimpleSales, we can navigate the syntax tree to access the part of the query that passes the parameter to the Sql.Database call.

## Value.Expression returns an *abstract syntax tree* (AST)



```
let
    Source = SimpleSales,
    Exp = Value.Expression(Value.Optimize(Source))
in
    Exp
```

Here, the top node of the tree of the optimized value expression is a function invocation.



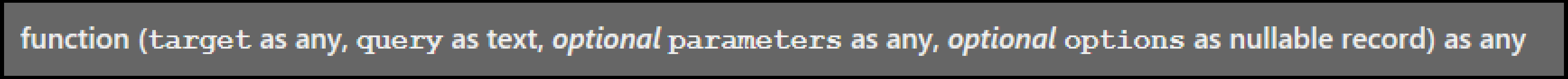
```
= Value.Expression(Value.Optimize(Source))
```

| | |
|---|---|
| Kind | Invocation |
| Function | Record |
| Arguments | List |

| | |
|---|---|
| Kind | Constant |
| Value | Function |

The expression record includes a 'Function' field, whose 'Value' field is the type of the function being invoked.

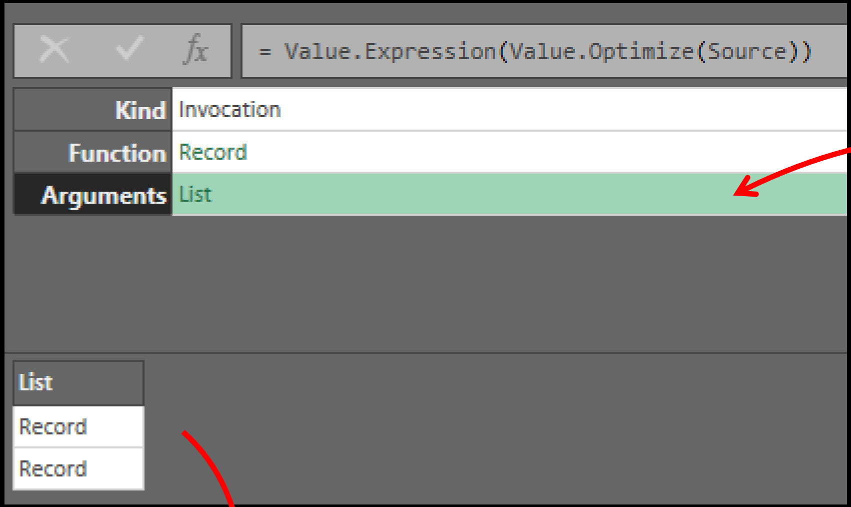
This is the function signature of Value.NativeQuery



function (target as any, query as text, *optional* parameters as any, *optional* options as nullable record) as any

## The function invocation includes the list of arguments being used

```
let
    Source = SimpleSales,
    Exp = Value.Expression(Value.Optimize(Source))
in
    Exp
```

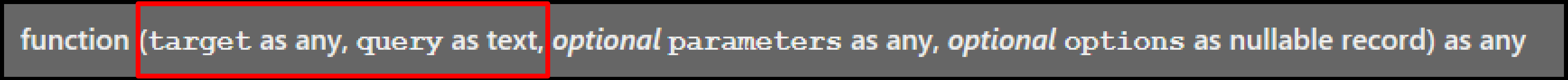


```
= Value.Expression(Value.Optimize(Source))
```

| | |
|---|---|
| Kind | Invocation |
| Function | Record |
| Arguments | List |

| List |
|---|
| Record |
| Record |

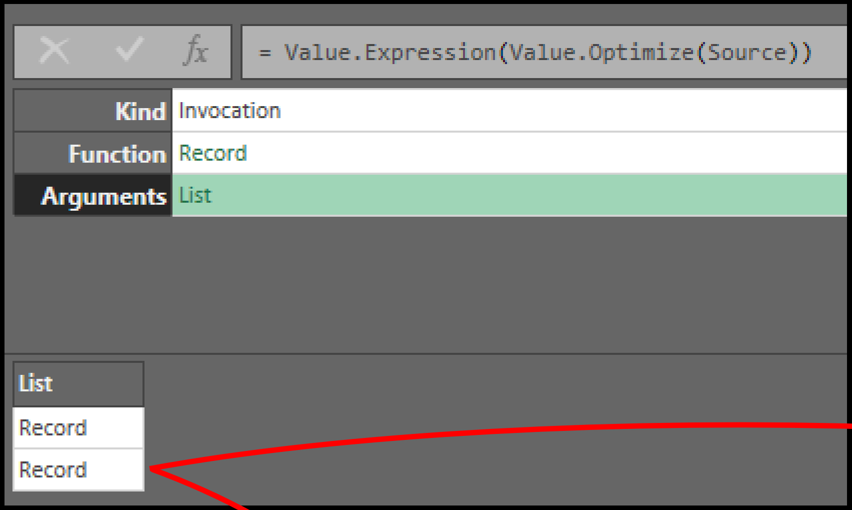
The Arguments field contains a list of the argument values passed to the function being invoked



function (target as any, query as text, *optional* parameters as any, *optional* options as nullable record) as any

# The second argument contains the SQL query

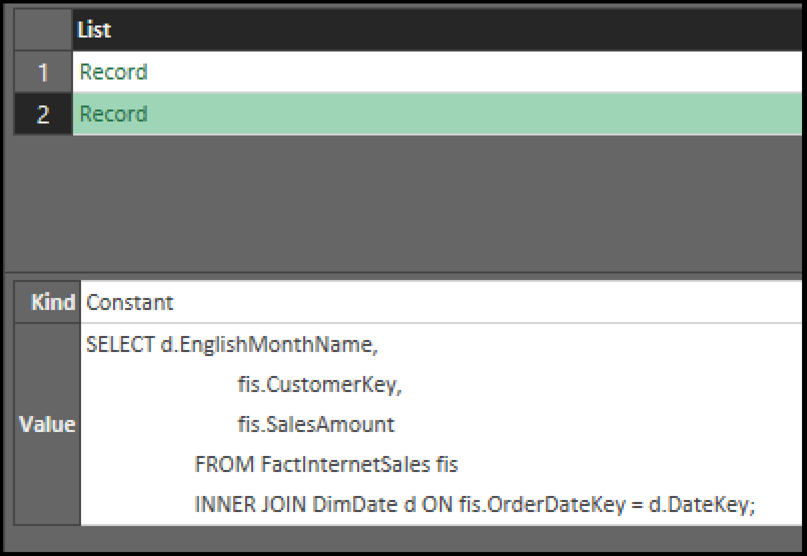
```
let
    Source = SimpleSales,
    Exp = Value.Expression(Value.Optimize(Source))
in
    Exp
```



```
= Value.Expression(Value.Optimize(Source))
```

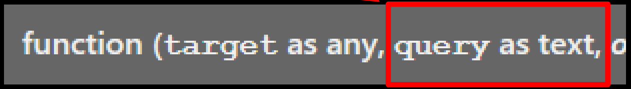
| | |
|---|---|
| **Kind** | Invocation |
| **Function** | Record |
| **Arguments** | List |

| List |
|---|
| Record |
| Record |



| | List |
|---|---|
| 1 | Record |
| 2 | Record |

| | |
|---|---|
| **Kind** | Constant |
| **Value** | SELECT d.EnglishMonthName, fis.CustomerKey, fis.SalesAmount FROM FactInternetSales fis INNER JOIN DimDate d ON fis.OrderDateKey = d.DateKey; |

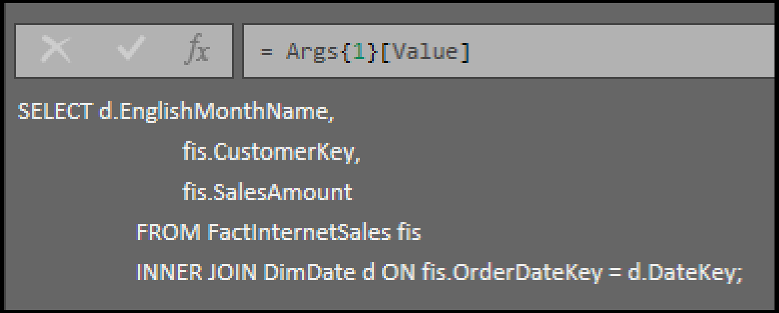


function (target as any, query as text, o

flexyourdata.com | youtube.com/@flexyourdata | linkedin.com/in/owenhprice

## So we can review the SQL text directly

```
let
    Source = SimpleSales,
    Exp = Value.Expression(Value.Optimize(Source)),
    Args = Exp[Arguments],
    SQL = Args{1}[Value]
in
    SQL
```

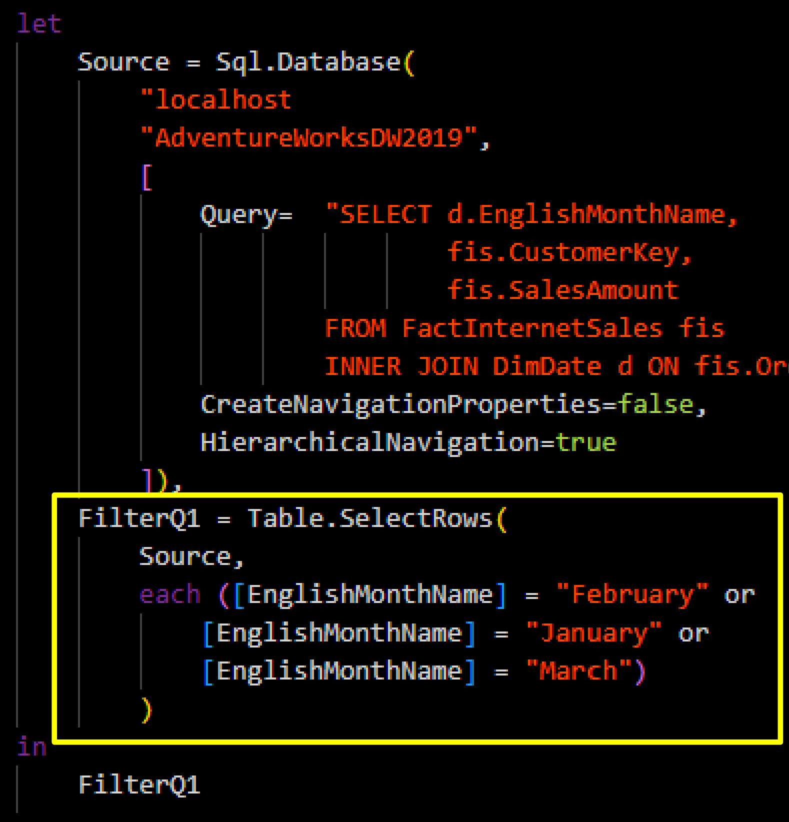


```
= Args{1}[Value]
```

```
SELECT d.EnglishMonthName,
          fis.CustomerKey,
          fis.SalesAmount
       FROM FactInternetSales fis
       INNER JOIN DimDate d ON fis.OrderDateKey = d.DateKey;
```

## OK. So what? ➔

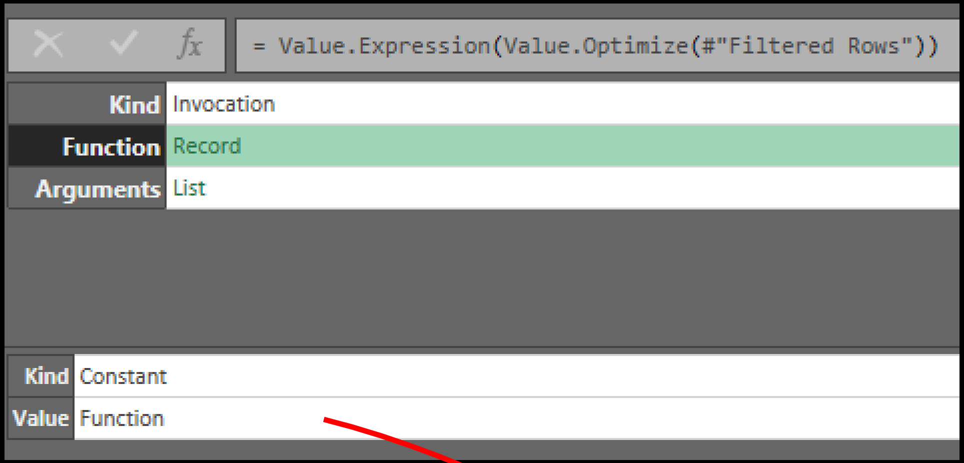
## If we add more steps, we can see whether they are being folded to the source



```
let
    Source = Sql.Database(
        "localhost
        "AdventureWorksDW2019",
        [
            Query=  "SELECT d.EnglishMonthName,
                            fis.CustomerKey,
                            fis.SalesAmount
                    FROM FactInternetSales fis
                    INNER JOIN DimDate d ON fis.OrderDateKey = d.DateKey;",
            CreateNavigationProperties=false,
            HierarchicalNavigation=true
        ]),
    FilterQ1 = Table.SelectRows(
        Source,
        each ([EnglishMonthName] = "February" or
            [EnglishMonthName] = "January" or
            [EnglishMonthName] = "March")
        )
in
    FilterQ1
```

This filters the data to only include January, February and March

# The AST for the SimpleSales query has now changed

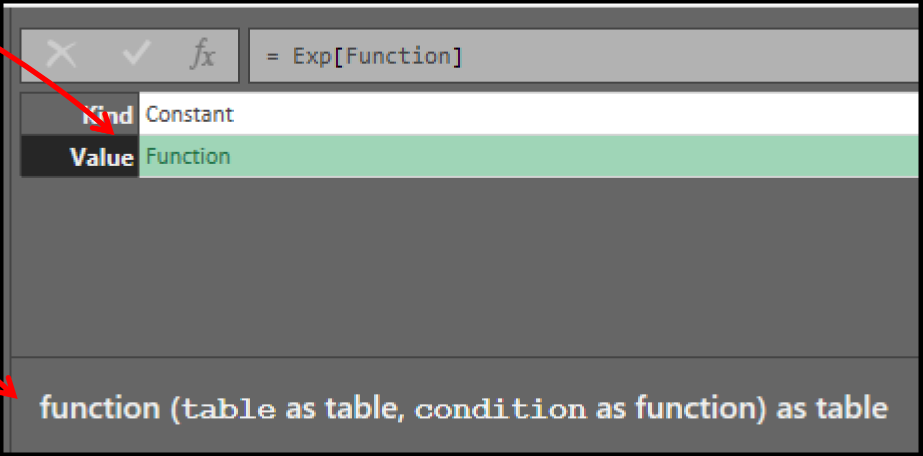


```
= Value.Expression(Value.Optimize(#"Filtered Rows"))
```

| | |
|---|---|
| Kind | Invocation |
| Function | Record |
| Arguments | List |

| | |
|---|---|
| Kind | Constant |
| Value | Function |

The top node is still a function invocation, but the function being invoked has changed



```
= Exp[Function]
```

| | |
|---|---|
| Kind | Constant |
| Value | Function |

function (table as table, condition as function) as table

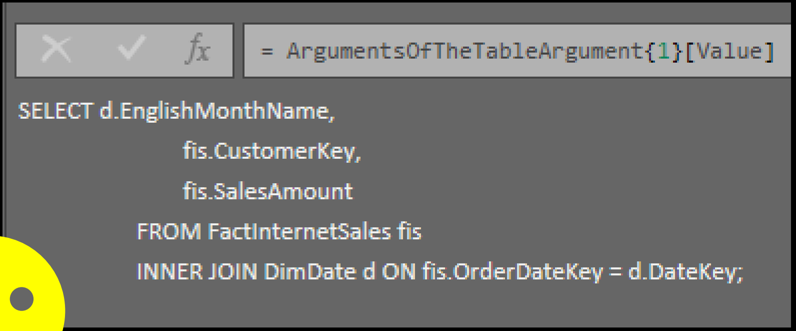
This is the function signature of Table.SelectRows

## We now need to dig further into the tree to find the SQL being passed to the database

```
let
    Source = SimpleSales,
    FilterQ1 = Table.SelectRows(
        Source,
        each (
                [EnglishMonthName] = "February" or
                [EnglishMonthName] = "January" or
                [EnglishMonthName] = "March")
        ),
    Exp = Value.Expression(Value.Optimize(FilterQ1)),
    ArgumentsOfTableSelectRows = Exp[Arguments],
    TheTableArgument = ArgumentsOfTableSelectRows{0},
    ArgumentsOfTheTableArgument = TheTableArgument[Arguments],
    SQL = ArgumentsOfTheTableArgument{1}[Value]
in
    SQL
```

The expression passed to the first parameter of Table.SelectRows is the invocation of Value.NativeQuery

The SQL going to the database does not include the WHERE clause – the filter transformation has *not* been folded



```
= ArgumentsOfTheTableArgument{1}[Value]
```

```
SELECT d.EnglishMonthName,
            fis.CustomerKey,
            fis.SalesAmount
        FROM FactInternetSales fis
        INNER JOIN DimDate d ON fis.OrderDateKey = d.DateKey;
```

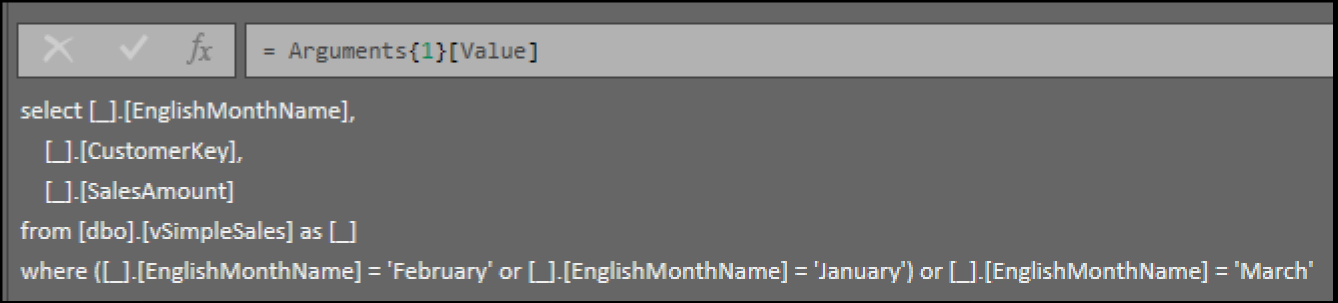
# This is because of the join in the base query

```
CREATE VIEW dbo.vSimpleSales
AS
SELECT d.EnglishMonthName, fis.CustomerKey, fis.SalesAmount
FROM FactInternetSales fis
INNER JOIN DimDate d ON fis.OrderDateKey = d.DateKey;
```

To enable more workload to happen on the SQL Server, we can create the base query as a view in the database

```
let
    Source = vSimpleSales,
    FilterQ1 = Table.SelectRows(
        Source,
        each (
              [EnglishMonthName] = "February" or
              [EnglishMonthName] = "January" or
              [EnglishMonthName] = "March")
        ),
    Exp = Value.Expression(Value.Optimize(FilterQ1)),
    Arguments = Exp[Arguments],
    SQL = Arguments{1}[Value]
in
    SQL
```

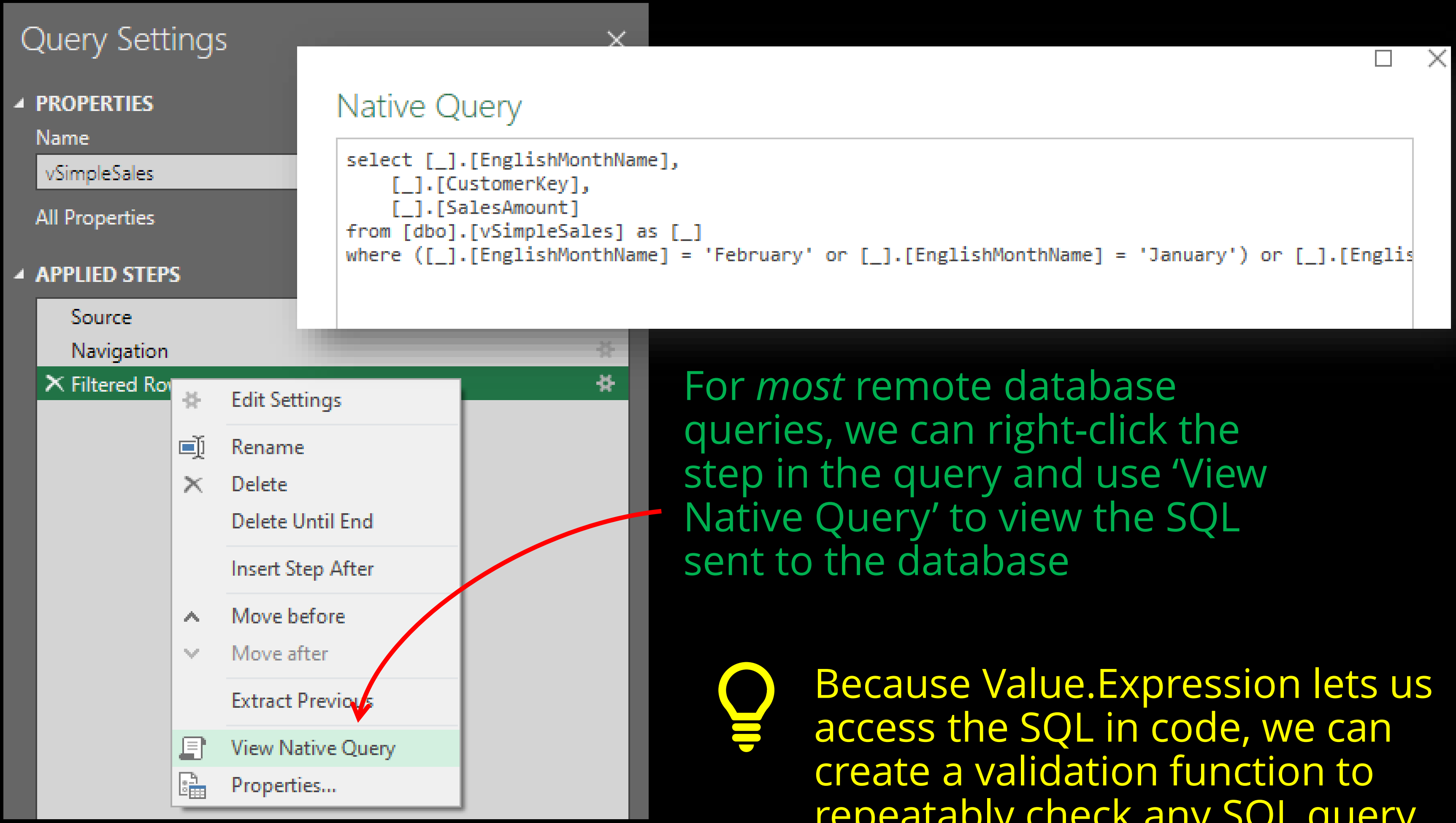
The filter transformation is now folded into the Value.NativeQuery call and the WHERE clause added to the SQL that it sent to the server



```
= Arguments{1}[Value]
```

```
select [_].[EnglishMonthName],
    [_].[CustomerKey],
    [_].[SalesAmount]
from [dbo].[vSimpleSales] as [_]
where ([_].[EnglishMonthName] = 'February' or [_].[EnglishMonthName] = 'January') or [_].[EnglishMonthName] = 'March'
```

# But wait… why don't I just use the UI?



```
select [_].[EnglishMonthName],
    [_].[CustomerKey],
    [_].[SalesAmount]
from [dbo].[vSimpleSales] as [_]
where ([_].[EnglishMonthName] = 'February' or [_].[EnglishMonthName] = 'January') or [_].[Englis
```

For *most* remote database queries, we can right-click the step in the query and use 'View Native Query' to view the SQL sent to the database

Because Value.Expression lets us access the SQL in code, we can create a validation function to repeatably check any SQL query step to see whether it's being folded or not.

## Takeaways

1. The Value.Expression function returns an *abstract syntax tree* – a way to navigate the steps of a query and the functions being executed in what order

2. One use of the AST is to programmatically access the SQL being sent to a remote database

3. This allows us to automatically inspect the state of query folding in complex mashups